Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Pasarela de la plataforma FIWARE al estándar médico FHIR basada en Spring

Autor: Jaime González Ruiz
Tutores: Isabel Román Martínez y Juan Antonio Ternero Muñiz

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Pasarela de la plataforma FIWARE al estándar médico FHIR basada en Spring

Autor:
Jaime González Ruiz

Tutores:
Isabel Román Martínez y
Juan Antonio Ternero Muñiz

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Grado: Pasarela de la plataforma FIWARE al estándar médico FHIR basada en Spring

Autor: Jaime González Ruiz

Tutores: Isabel Román Martínez y
Juan Antonio Ternero Muñiz

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia
A mis amigos

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres su apoyo incondicional, tanto emocional como económico, a lo largo de estos intensos cuatro años. Agradecer también a mis amigos y a mis compañeros por hacer estos años más llevaderos. Gracias por las risas y los momentos vividos que quedarán ahí para siempre. Pero sobre todo agradecer a mis tutores Isabel y Juan Antonio, que me han orientado cuando lo he necesitado y me han ayudado en la revisión de este documento.

Jaime González Ruiz
Sevilla, 2019

Resumen

FIWARE [1] es una plataforma para IoT que, aunque ha sido utilizada en dominios como las ciudades inteligentes, la industria o la gestión de la energía, aún se ha utilizado poco en el contexto de la sanidad.

Un problema recurrente en el ámbito de las tecnologías enfocadas a la gestión de la salud es la interoperabilidad entre ellas. En los últimos años, se ha observado una tendencia al alza en la inversión en las TICS para crear una arquitectura común, que permita entre otras cosas el intercambio de mensajes entre las aplicaciones sanitarias, la interoperabilidad del Historial Médico Electrónico y los identificadores de los pacientes.

El estándar FHIR [2], impulsado por HL7 [3], es un proyecto cuyo objetivo es proporcionar una arquitectura moderna, interoperable y común para las aplicaciones del ámbito médico, solucionando el enorme problema de la comunicación entre sistemas.

Este Trabajo de Fin de Grado consiste en la creación de una aplicación web que actúa como pasarela entre dispositivos médicos de la plataforma FIWARE y el estándar FHIR. Para alcanzar este objetivo, se ha establecido una correspondencia entre los recursos de ambos y se han desarrollado una serie de funciones tras una interfaz REST [4] para manejar arquitecturas que usen ambos estándares.

Abstract

FIWARE is an IoT focused platform. It has been extensively used in well-known domains such as smart cities, the industry or energy management. However, it has been barely used in the eHealth environment.

A recurrent problem in the eHealth focused technologies is the interoperability among them. In the last years, there has been an increasing investment in ICTS to create a common architecture. This would enable the exchange of messages among medical applications, the interoperability of the Electronic Medical Record (EMR) and patients' identifiers.

FHIR is a standard for electronic exchange of healthcare information developed by HL7. FHIR is a project which goal is to offer a modern, interoperable and common architecture for medical applications. It solves the huge problem of communication among systems.

This thesis is focused in the creation of a web application that acts has a gateway between medical devices in FIWARE and the FHIR standard. In order to achieve this goal, I have established a mapping among the resources of both frameworks, and I have developed a series of functions behind a REST interface to manage architectures using both standards.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Introducción

Gracias al avance de la tecnología cada vez contamos con sensores médicos más avanzados que permiten la teleasistencia y la monitorización del paciente de forma remota en tiempo real. Estos sensores son capaces de comunicarse con el exterior tanto por medios cableados como de forma inalámbrica, ya que poseen conexiones como Wifi, Bluetooth o Zigbee. Generalmente, no son capaces de conectarse a Internet por sí solos, por lo que lo hacen a través de una pasarela que puede ejecutarse en un ordenador, un smartphone o una placa Raspberry Pi o Arduino.

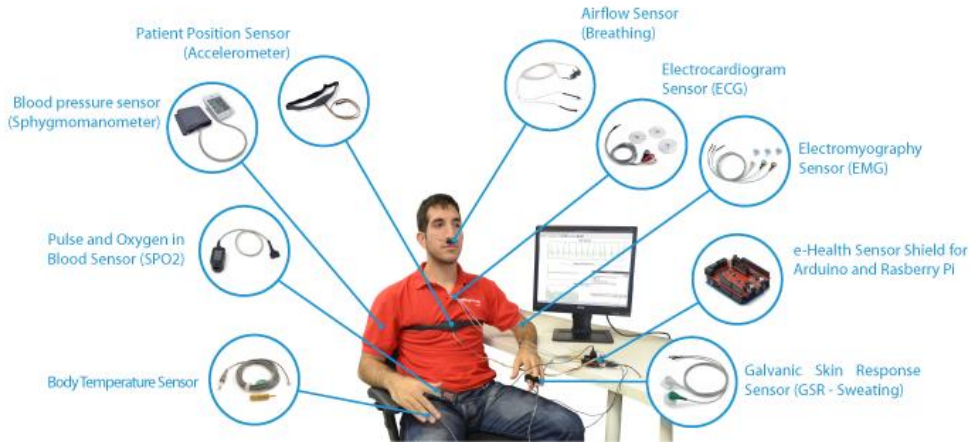

Figura 1. Sensores médicos conectados a Internet a través de una Raspberry Pi

FIWARE [1] es una iniciativa open source (código abierto en español) que pretende impulsar la creación de estándares necesarios para desarrollar aplicaciones en diferentes dominios: Smart Cities, Smart Ports, Smart Logistics, Smart Factories, entre otros. Cualquier aplicación basada en FIWARE, se caracteriza por recoger información relevante sobre lo que está pasando en un momento dado, procedente de diferentes fuentes. Esto se conoce como "información de contexto". La información de contexto actual e histórica se procesa, visualiza y analiza a gran escala. De esta forma, se produce el comportamiento inteligente esperado.

FIWARE quiere impulsar un estándar que describe cómo recopilar, gestionar y publicar información de contexto y adicionalmente aporta elementos que permiten explotar esta información una vez recopilada. Este estándar resulta clave para construir un mercado digital único para las aplicaciones inteligentes donde las apps y soluciones pueden ser utilizadas por distintos clientes sin grandes cambios. También resuelve de manera sencilla cómo capturar información procedente de redes de sensores, que se comunican usando diferentes protocolos y lenguajes IoT. En ese sentido es capaz de resolver la complejidad de tratar la información recogida por los sensores y traducirlos a un lenguaje común.

FIWARE proporciona una serie de servicios denominados enablers, que proveen prestaciones

muy reutilizables en el dominio IoT. El servicio central de FIWARE lo proporciona el Orion Context Broker [5]. Orion es una implementación en C++ de la API REST NGSIv2 [6] que veremos en profundidad más adelante. Es una pieza fundamental de la plataforma FIWARE. Usando este enabler es posible registrarse a elementos del contexto y administrarlos a través instrucciones de consulta y actualización. Además, permite la suscripción a los datos capturados por los sensores, y una vez ocurran las condiciones establecidas, se enviará una notificación.

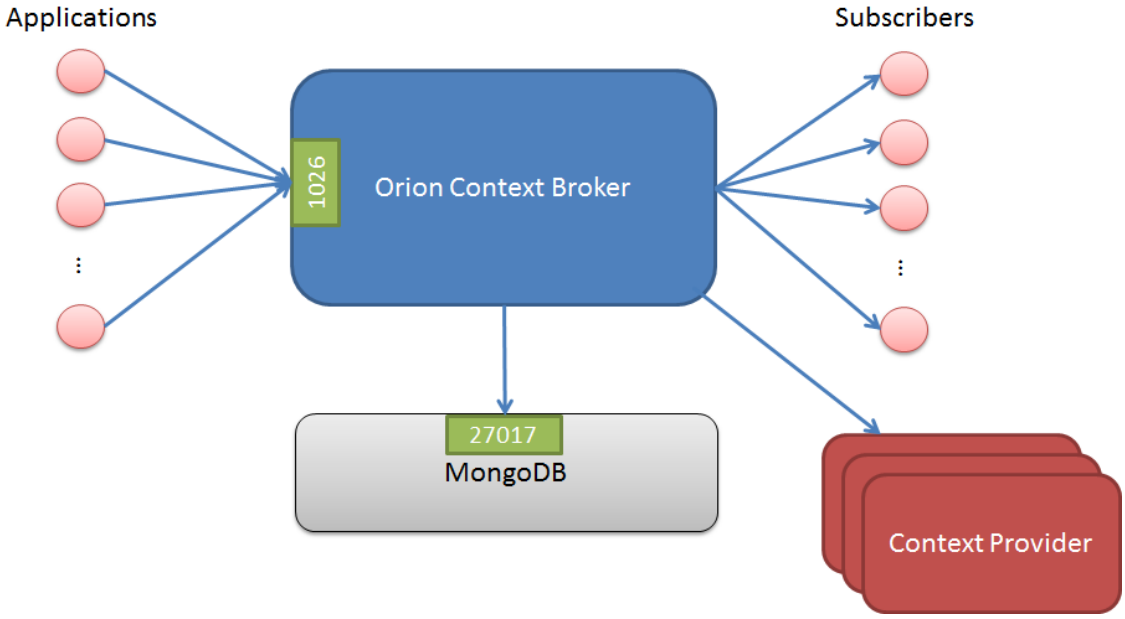

Figura 2. Orion Context Broker

Se requiere un componente dentro de la arquitectura que sea capaz de mediar entre los productores de datos (en este caso los sensores médicos) y los consumidores de datos (como la pasarela basada Java Spring que aprovecha la información proveída por los sensores). Orion Context Broker satisface estas funcionalidades en la arquitectura de FIWARE.

El Orion Context Broker utiliza una base de datos MongoDB [7], donde se guarda toda la información. El broker almacena los datos de las entidades y de las suscripciones en la base de datos. Además, el Broker y la base de datos no tienen por qué estar en la misma máquina, lo que otorga una mayor flexibilidad. Además, podríamos tener varias instancias de la base de datos o del Broker para asegurar cierta redundancia y protegernos de los fallos más críticos.

FHIR [2] o Fast Healhcare Interoperability Resources, es un estándar de interoperabilidad para el intercambio de recursos electrónicos sobre información sanitaria. FHIR fue desarrollado por HL7 [3], una organización sin ánimo de lucro acreditada por el Instituto Americano Nacional de Estándares, la cual desarrolla y proporciona estándares y entornos para la integración, el intercambio y la recuperación de datos clínicos y otros datos relacionados con la salud.

Hoy en día, los sistemas de información sanitarios continúan estando plagados de problemas de interoperabilidad. FHIR emergió en 2014 como un borrador en estado de pruebas, para permitir a los desarrolladores construir de forma más fácil y rápida aplicaciones relacionas con los historiales médicos electrónicos, y recuperar e intercambiar información médica de forma más fácil y estandarizada.

Cuando empezó en 2014, FHIR no era más que un experimento para HL7, pero pronto fue apoyado por grandes empresas y grupos americanos. Esto llevó a FHIR al punto de que, en febrero de 2017, se convirtió en un estándar completo.

FHIR sigue el paradigma REST [4], está construido entorno a una serie de unidades básicas de interoperabilidad denominadas recursos. Existen recursos como pacientes, proveedores, organizaciones y dispositivos, además de una variedad de conceptos clínicos como pueden ser medicaciones, diagnósticos y tipos de cuidados médicos entre otros. Este estudio particular se centra en 4 recursos concretos: Device [8], DeviceDefinition [9], DeviceMetric [10] y Observation [11].

La idea de este proyecto es desarrollar una aplicación intermedia o pasarela entre una aplicación basada en FIWARE y un servidor que sigue el estándar médico FHIR. La pasarela crea una serie de suscripciones en Orion para que sólo nos notifique cuando cambian los valores clave de 5 atributos. Estos atributos son el valor medido por el sensor, el estado del sensor, su dirección IP, su posición GPS y la última fecha de calibración. El resto de los atributos, como se verá más adelante, tiene un carácter más estático y no van a ser cambiados en la operación normal del sensor.

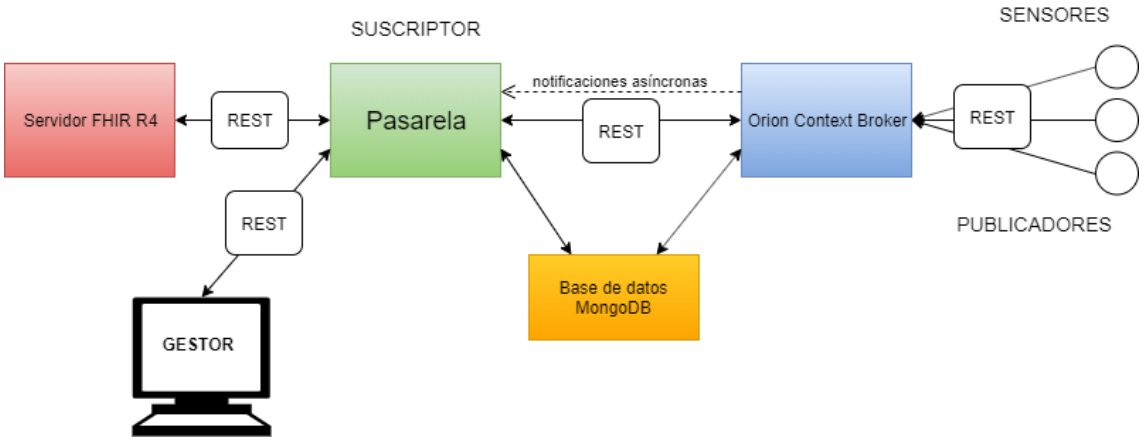

Figura 3. Relaciones de la aplicación con el resto de los componentes

La pasarela ofrece dos interfaces. Una interfaz REST es utilizada por el gestor para dar de alta, modificar y borrar dispositivos. En la segunda interfaz están los puntos de entrada que procesan las notificaciones HTTP [12] del Context Broker. Además, se utilizan dos clientes REST. Uno para comunicarse con el Orion Context Broker y el otro con el servidor FHIR.

Una familia de dispositivos representa una caracterización de un tipo de dispositivo y uno o más dispositivos físicos que pertenecen a este tipo concreto. Una familia de dispositivos presenta como mínimo uno de cada uno de los recursos FHIR con los que trabaja la aplicación. Estos recursos son Device, DeviceDefinition, DeviceMetric y Observation. Con estos cuatro recursos se construye una familia con un único dispositivo. Para añadir más dispositivos basta con añadir más recursos Device, los otros tres recursos son comunes para toda la familia.

Para dar de alta a una familia de dispositivos, el administrador o gestor tiene que mandar una serie de plantillas en formato JSON [13] que siguen el estándar FHIR. A partir de estas plantillas, se crean las entidades correspondientes en el sistema FIWARE. Mediante estas plantillas con información estática del dispositivo y los valores que se reciben en las notificaciones HTTP de Orion se crean recursos completos y se envían al servidor FHIR.

1.2 Objetivos

- Establecer una correspondencia entre los recursos del estándar FIWARE y FHIR relacionados con los dispositivos en cada plataforma.
- Desarrollo de una aplicación web que actúe como pasarela.
- Modelado de las suscripciones y entidades FIWARE en java.
- Modelado de los recursos asociados a dispositivos de FIWARE en java.

- Desarrollo de una interfaz REST para gestionar la aplicación.
- Desarrollo de una interfaz REST para reaccionar ante las notificaciones de Orion.
- Desarrollo de un cliente REST para la comunicación con el Context Broker.
- Desarrollo de un cliente REST para la comunicación con el servidor FHIR R4.
- Desarrollo de un cliente para la comunicación con la base de datos MongoDB.
- Despliegue de un servidor FHIR R4.
- Despliegue de una base de datos MongoDB.
- Despliegue del Orion Context Broker.
- Despliegue de la pasarela sobre Tomcat.
- Pruebas.

1.3 Herramientas

1.3.1 Eclipse Java EE IDE for Web Developers

Eclipse [14] es un entorno de desarrollo integrado (IDE, en inglés) usado típicamente para el desarrollo de aplicaciones. Ofrece un editor de texto y herramientas para ejecutar y depurar el código de forma ágil y cómoda. Además, puede integrar el contenedor de servlets Tomcat y Maven. Maven se encarga de resolver todas las dependencias del código, además ofrece herramientas para empaquetar la aplicación a un jar con todo lo necesario para su ejecución de forma independiente. Eclipse también incluye un explorador de archivos y una consola donde podemos ver la ejecución de nuestra aplicación.

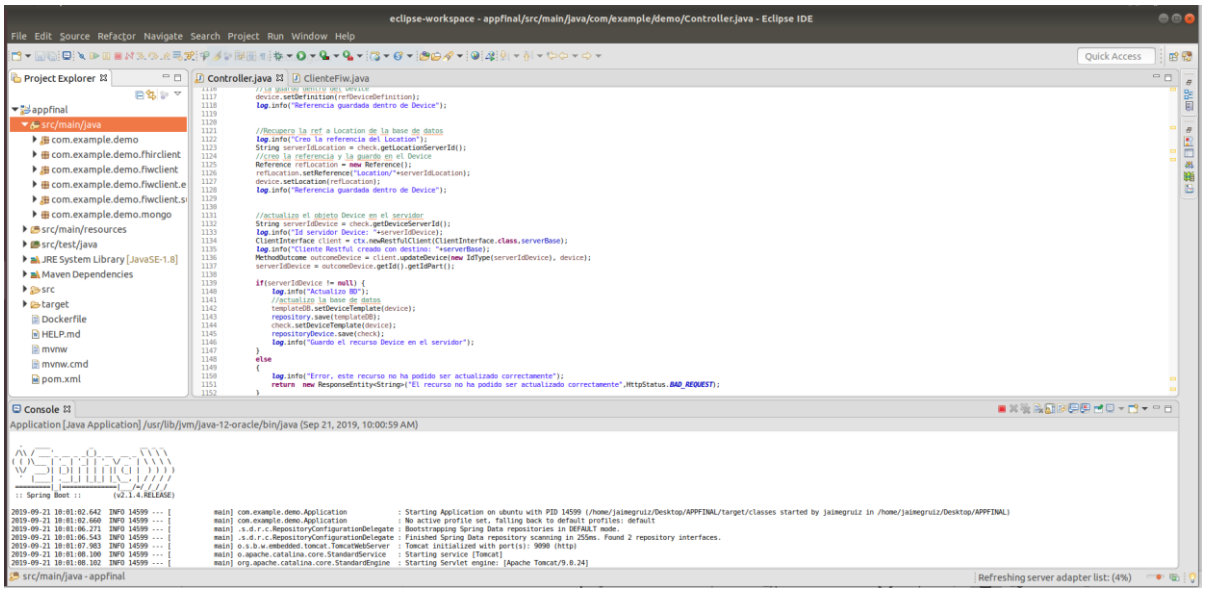

Figura 4. Aplicación Eclipse

1.3.2 Postman

Postman [15] es una herramienta que permite hacer pruebas sobre interfaces RESTful de forma muy sencilla. Ofrece una interfaz de usuario muy intuitiva para fabricar peticiones HTTP de todos los tipos.

Para crear una nueva petición, solo hay que seleccionar el tipo del mensaje HTTP y especificar la URL destino. Además, se pueden añadir parámetros a la URL, cabeceras HTTP al mensaje y

especificar su cuerpo si es necesario. Una vez el mensaje está fabricado se envía pulsando un botón, y abajo aparece la respuesta a la petición totalmente diseccionada.

Se ha usado esta aplicación tanto para probar las funcionalidades de la interfaz REST de la pasarela, como para simular las actualizaciones en los valores de los sensores. Además, se ha creado una colección con todas las peticiones que se utilizan normalmente y es posible compartirla fácilmente a otros desarrolladores.

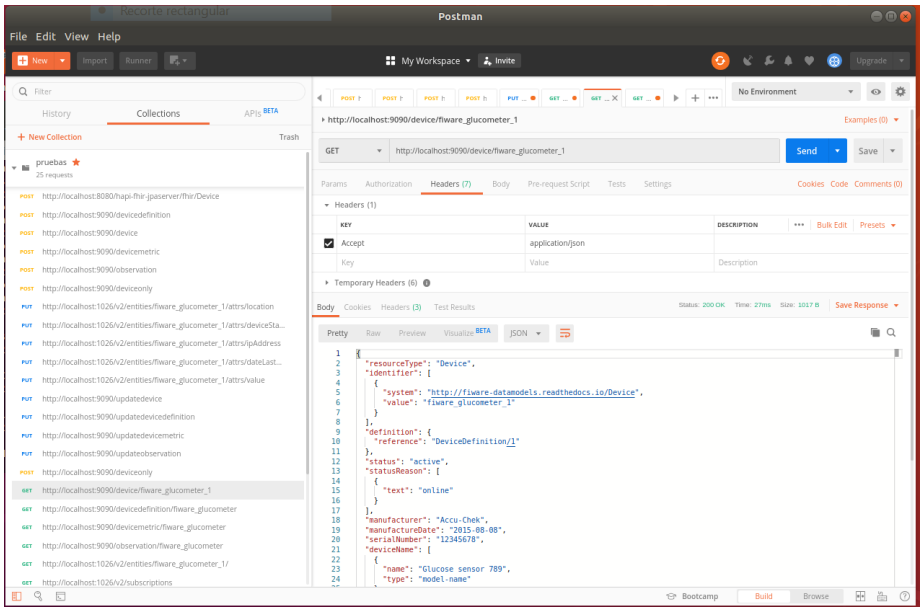

Figura 5. Aplicación Postman

1.3.3 VMware Workstation 15 Player

VMware Workstation 15 Player [16] es una herramienta gratuita para ejecutar una máquina virtual en un equipo con Windows o Linux. Se ha usado para lanzar una máquina virtual con la versión de Ubuntu 18.04.3 LTS sobre Windows 10. El desarrollo resulta más cómodo en Ubuntu, además se puede compartir la máquina virtual con todo el trabajo montado listo para funcionar.

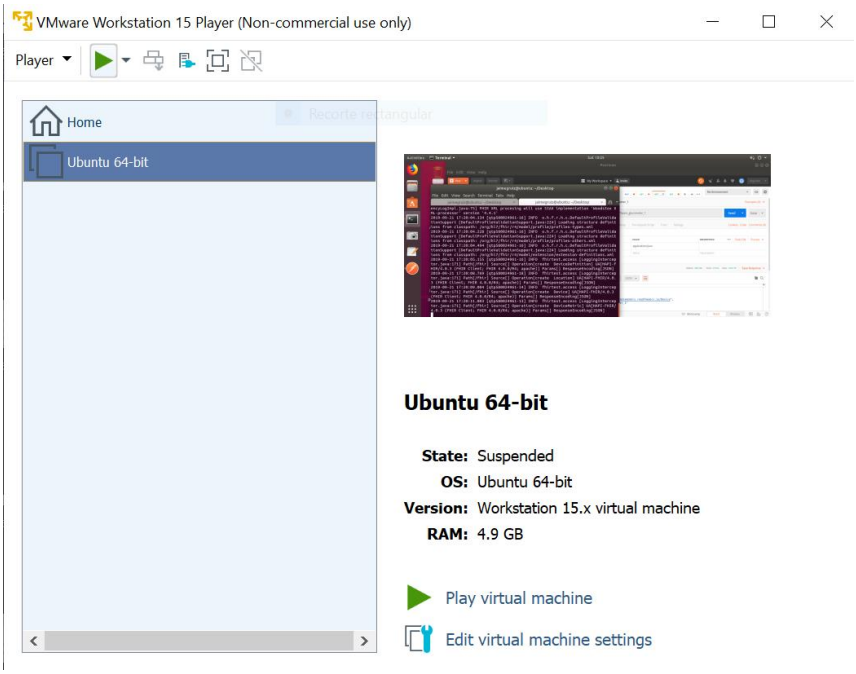

Figura 6. Aplicación VMware Workstation 15 Player

1.3.4 draw.io

draw.io [17] es una aplicación web gratuita, muy integrada con los servicios de almacenamiento en la nube de Google Drive o OneDrive, que permite dibujar diagramas de todo tipo completamente online. Se ha usado a lo largo de este trabajo para crear algunos diagramas de apoyo a las explicaciones.

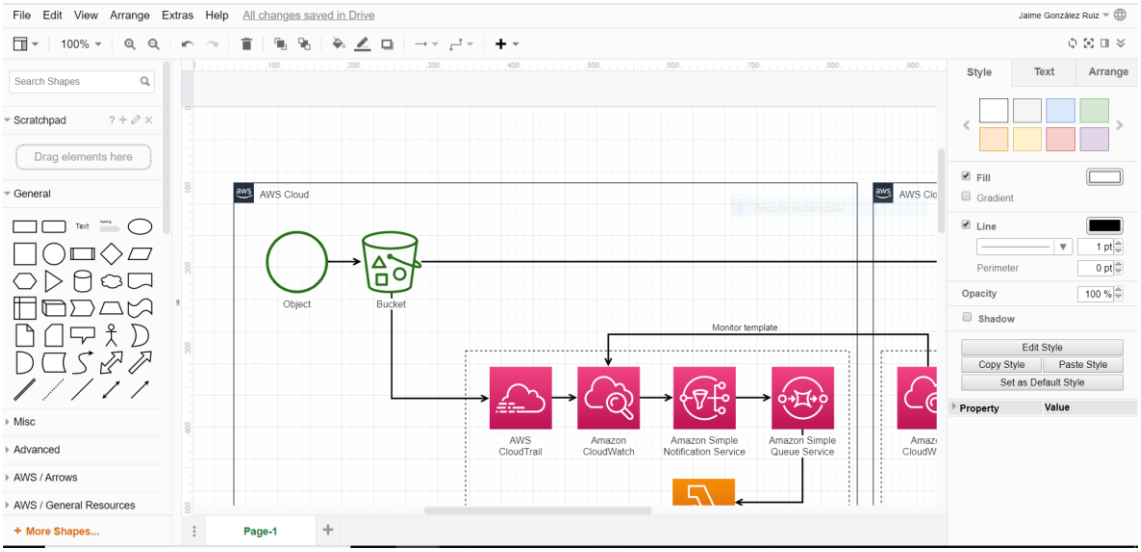

Figura 7. Aplicación draw.io

1.4 Descripción de la memoria

Una vez que se ha terminado de introducir el proyecto, se va a explicar en detalle las tecnologías que lo han hecho posible en el capítulo 2. En el capítulo 3 se va a profundizar en el trabajo realizado, tanto en la correspondencia entre los recursos de FHIR y FIWARE como en la aplicación java en sí. Explicando sus diferentes partes, el cliente FIWARE, el cliente FHIR, el cliente de la base de datos MongoDB y el controlador REST.

En el capítulo 4 se va a explicar cómo se ha montado el escenario completo y se van a realizar pruebas a cada una de las funciones que ofrece la interfaz REST. Finalmente, en el capítulo 5 se enunciarán las conclusiones del proyecto.

2 ESTADO DE LA TÉCNICA

2.1 Java

Java [18] es un lenguaje de programación de propósito general, concurrente, basado en clases y orientado a objetos. Además, está diseñado para tener el menor número de dependencias de implementación posible.

El lenguaje Java, está pensado para que los desarrolladores escriban el programa una sola vez y que lo ejecuten en cualquier dispositivo. Esto significa que el código Java compilado puede ser ejecutado en todas las plataformas que soporten java sin necesidad de ser recompilado. Todo esto es posible porque las aplicaciones de Java se compilan a bytecode que puede ser ejecutado en cualquier máquina virtual de Java (JVM), independientemente de la arquitectura del ordenador que lo ejecuta.

Hoy en día, Java es uno de los lenguajes de programación más populares. Se usa principalmente para aplicaciones web cliente-servidor, pero también lo encontramos en el desarrollo de aplicaciones de escritorio y aplicaciones móviles.

La elección de Java para realizar este proyecto no se debe solo a su gran popularidad. Es un lenguaje ya conocido, lo que disminuye la curva de aprendizaje inicial para el desarrollo del proyecto. Además, la librería HAPI FHIR [19], que es uno de los ejes de este proyecto, está escrita en este lenguaje. Si a todo esto, le añadimos la versatilidad de frameworks como Spring [20], está claro que Java es una buena elección para este proyecto.

2.2 JSON

JSON [13] (JavaScript Object Notation) es un formato de intercambio de datos basado en texto que deriva del lenguaje JavaScript. Normalmente, se usa en servicios web y otras aplicaciones. Es fácil de leer y escribir para los humanos, pero también es fácil de generar y diseccionar para las máquinas. Además, es un formato de texto totalmente independiente del lenguaje de programación utilizado.

En JSON solo se definen dos estructuras de datos, objetos y arrays. Un objeto es un conjunto de pares nombre-valor, y un array es una lista de objetos. Además, solo se definen 7 tipos de valores posibles: string, número, objeto, “array”, true, false y null.

El siguiente ejemplo se observa un objeto JSON que contiene algunos pares nombre-valor. El valor para “phoneNumbers” es una lista que contiene dos objetos en su interior.

```
{
   "firstName": "Duke",
   "lastName": "Java",
   "age": 18,
   "streetAddress": "100 Internet Dr",
   "city": "JavaTown",
   "state": "JA",
   "postalCode": "12345",
   "phoneNumbers": [
      { "Mobile": "111-111-1111" },
      { "Home": "222-222-2222" }
   ]
}
```

Figura 8. Objeto JSON

La plataforma FIWARE usa JSON para intercambiar información de contexto. Así como para intercambiar información sobre las suscripciones y en el cuerpo de los mensajes de notificación. En cuanto a FHIR, son válidos tantos los formatos JSON y XML. Al usarse JSON en FIWARE, se ha optado por representar los recursos FHIR y trabajar con ellos en formato JSON. Además, en la base de datos MongoDB se almacenan documentos JSON con información esencial para el funcionamiento de la aplicación. Por lo tanto, los 3 bloques del trabajo, FIWARE, FHIR y la pasarela, usan JSON para transmitir información entre ellos.

2.3 MongoDB

MongoDB [7] es una base de datos no relacional de documentos que ofrece una gran escalabilidad, flexibilidad y modelo de consultas e indexación avanzado. Los datos son almacenados en documentos JSON flexibles, es decir, cada documento puede contener diferentes campos y las estructuras de datos se pueden ir modificando. El modelo de documentos concuerda con los objetos del código de la aplicación, lo que facilita trabajar con la información. Las consultas ad-hoc, la indexación y la agregación en tiempo real permiten acceder a los datos y analizarlos con gran eficacia. MongoDB es una base de datos distribuida, fácil de usar y proporciona una elevada disponibilidad, escalabilidad horizontal y distribución geográfica.

El modelo de documentos de MongoDB resulta muy fácil de aprender y usar, y proporciona a los desarrolladores todas las funcionalidades que necesitan para satisfacer los requisitos más complejos a cualquier escala. Se proporcionan drivers para más de diez lenguajes, y la comunidad ha desarrollado varias decenas más. Grandes empresas como Amazon, Cisco y Adobe confían y utilizan MongoDB en sus productos.

Como se ha comentado, Orion necesita una base de datos MongoDB para almacenar la información. Además, la pasarela necesitaba una base de datos y MongoDB facilitaba mucho el trabajo comparado con una alternativa tradicional de tipo SQL. FIWARE y FHIR usan JSON para transmitir la información. Por lo tanto, la solución basada en plantillas se beneficia enormemente de poder almacenar directamente documentos JSON que pueden ser modificados y enviados de forma fácil y rápida. Finalmente, el framework Spring de Java hace que usar MongoDB sea más fácil todavía.

2.4 HTTP

HTTP [12] siglas de Hypertext Transfer Protocol, es un protocolo de nivel de aplicación sin estado, usado en sistemas de información distribuidos, colaborativos y de hipertexto. En la RFC 7231 se puede encontrar la definición de la semántica de los mensajes HTTP/1.1, profundizando en los métodos de petición, en los campos de cabecera en la petición, en los códigos de estado de la respuesta y en los campos de cabecera de la respuesta. Así como en el cuerpo de los mensajes y los mecanismos para la negociación del contenido.

HTTP es el protocolo a nivel de aplicación que permite que las piezas del sistema se comuniquen entre sí.

2.5 Servicio web RESTful

REST [4] equivale a "Representational State Transfer" en inglés, es un estilo arquitectural de software que especifica una serie de restricciones. Al aplicarse a un servicio web, inducen propiedades deseables como pueden ser el rendimiento, la escalabilidad, y la capacidad de cambio que permiten que los servicios web funcionen mejor. Un servicio web que implementa una arquitectura REST se describe como RESTful.

En la arquitectura REST, los datos y la funcionalidad son considerados recursos y son accedidos mediante URIs (Uniform Resource Identifiers), normalmente enlaces web tradicionales. La arquitectura REST obliga a usar una arquitectura cliente-servidor y está diseñada para usar un protocolo de comunicación sin estado, típicamente HTTP. Los clientes y los servidores intercambian representaciones de recursos usando una interfaz y protocolo estandarizados.

Los siguientes principios hacen que las aplicaciones RESTful sean simples, ligeras y rápidas:

- **Identificación de recursos mediante URI**: un servicio web RESTful expone un conjunto de recursos que identifican los objetivos de la interacción con sus clientes. Los recursos se identifican mediante URIs, que proporcionan un espacio de direccionamiento global para el descubrimiento de recursos y servicios.

- **Interfaz uniforme**: los recursos se manipulan usando un conjunto fijado de 4 operaciones de lectura, escritura, actualización y borrado. Mediante las operaciones PUT, GET, POST y DELETE. PUT puede tanto crear como actualizar un recurso si este ya existía con anterioridad. El recurso puede ser después borrado por DELETE. GET recupera el estado actual del recurso en alguna representación. POST también puede ser usado tanto para crear como actualizar recursos. Esto lleva a confusión con el método PUT. La respuesta rápida sería que POST es para crear y PUT para actualizar recursos. La principal diferencia es que el método PUT es idempotente, esto significa que producirá el mismo resultado si se ejecuta varias veces. POST no lo es, ya que al ejecutarlo varias veces estás creado varios elementos. Normalmente se sigue lo siguiente:

    - POST se utiliza para crear recursos y PUT para actualizar recursos ya existentes
    - PUT se usa cuando se conocen los identificadores de los recursos
    - POST se utiliza cuando es el servidor el que controla la generación de identificadores y URLs de los recursos
    - Como ejemplo:
        - PUT/ítems/1/update
        - POST/ítems/create

- **Mensajes auto descriptivos**: los recursos están desacoplados de su representación para que su contenido pueda ser accedido en una variedad de formatos, como HTML, XML, texto plano, PDF, JPEG, JSON entre otros. Los metadatos de los recursos existen y son usados para, por ejemplo, detectar errores de transmisión, negociar el formato adecuado de presentación y para realizar control de acceso o autenticación.

- **Interacciones con estado mediante hiperenlaces**: cada interacción con un recurso es sin estado, esto quiere decir que los mensajes de petición son auto contenidos. Las interacciones con estado se basan en el concepto explícito de transferencia de estado. Existen una serie de técnicas para la transferencia de estado, como pueden ser la sobreescritura de URI, las cookies, y campos ocultos en formularios. Estas técnicas permiten, por ejemplo, mantener el estado de la sesión de un usuario sobre HTTP que es un protocolo sin estado.

Tanto FIWARE como FHIR definen una interfaz REST. Además, la pasarela Java también define su propia interfaz REST que permite al gestor crear, obtener, actualizar y borrar familias de dispositivos y dispositivos concretos.

2.6 Spring Framework

Spring [20] es un framework de código abierto que se utiliza para el desarrollo de aplicaciones Java. Un framework en programación es el resultado de la evolución de la ingeniería del software, estos son creados por programadores para programadores, con la finalidad de estandarizar el trabajo, resolver, agilizar y manejar los problemas y complejidades que van apareciendo. Creando así, en la comunidad de desarrolladores, un abanico de posibilidades para una creación cada vez más evolucionada de aplicaciones.

Spring permite desarrollar aplicaciones de manera más rápida y eficaz, eliminando tareas repetitivas y ahorrando líneas de código. Fue escrito originalmente para la plataforma J2EE de Java, plataforma orientada al desarrollo de aplicaciones web y ha ido evolucionando rápidamente hasta el día de hoy, donde podemos encontrar diferentes ramas de desarrollo de la mano de SpringSource y todo su equipo de desarrolladores.

Spring en la actualidad es la referencia en el mundo de los frameworks de programación para los desarrolladores web de todo el mundo. Su éxito se fundamenta en la constante labor de investigación e innovación que realiza su equipo de desarrollo.

Spring fue una elección fácil ya que ofrecía librerías de gran utilidad para afrontar varios problemas del proyecto. Además, es uno de los frameworks más populares y recomendados del lenguaje Java. Ha ayudado, entre otras cosas, a lanzar de forma sencilla una aplicación web sobre Tomcat, ha facilitado la programación de un cliente REST, la interacción con la base de datos MongoDB y la creación de un controlador REST.

2.6.1 Spring Boot

Spring Boot [21] es un proyecto construido sobre el mencionado Spring Framework. Proporciona una forma más fácil y rápida de construir y ejecutar aplicaciones autónomas y basadas en la web. Dentro del Spring Framework fundamental, hay que configurar muchas cosas a mano mediante archivos XML, esto es una de las cosas que Spring Boot facilita.

Al iniciar un proyecto Java, el desarrollador tiene que tomar una serie de decisiones, como el framework a usar, la herramienta de logging, el servidor web, etc. No obstante, al final la mayoría de los desarrolladores usan las mismas herramientas una y otra vez. Lo que hace Spring Boot es

que carga y configura todas estas herramientas de la forma más genérica posible. Así, los desarrolladores no tienen que configurar lo mismo una y otra vez. Obviamente, esta configuración inicial puede ser posteriormente personalizada por el desarrollador, para satisfacer sus necesidades.

En este proyecto se ha usado Spring Boot principalmente para el lanzamiento y despliegue de la aplicación web, ya que incluye un servidor web Tomcat embebido. Además, Spring Boot analiza automáticamente los componentes de la pasarela y realiza las configuraciones necesarias. En este caso un componente @RestController, para crear la interfaz REST. Además, configura también de forma automática la conexión a la base de datos MongoDB.

2.7 Maven

Maven [22] es una herramienta software para la gestión y construcción de proyectos Java. Maven utiliza un Project Object Model (POM), que consiste en un documento XML que describe el proyecto software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación de código y su empaquetado.

Una característica principal de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el cual provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache, y otras organizaciones y desarrolladores. Este repositorio y su sucesor reorganizado, el repositorio Maven 2, pugnan por ser el mecanismo de facto de distribución de aplicaciones en Java.

Las librerías de Spring usan archivos de configuración POM que resuelven todas las dependencias necesarias y facilitan inmensamente el trabajo. Además, el uso de herramientas como Maven, o similares como puede ser Gradle, es de gran necesidad en cualquier proyecto Java de cierta complejidad. Este es un fragmento del archivo de configuración POM de la pasarela:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>appfinal</artifactId>
    <version>0.1.0</version>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.4.RELEASE</version>
    </parent>

     <dependencies>
         <dependency>
            <groupId>ca.uhn.hapi.fhir</groupId>
            <artifactId>hapi-fhir-base</artifactId>
            <version>4.0.3</version>
        </dependency>
        <dependency>
            <groupId>ca.uhn.hapi.fhir</groupId>
            <artifactId>org.hl7.fhir.utilities</artifactId>
            <version>4.0.3</version>
        </dependency>
        <dependency>
            <groupId>ca.uhn.hapi.fhir</groupId>
            <artifactId>hapi-fhir-client</artifactId>
            <version>4.0.3</version>
        </dependency>
        <dependency>
            <groupId>ca.uhn.hapi.fhir</groupId>
            <artifactId>hapi-fhir-structures-r4</artifactId>
            <version>4.0.3</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
```

Figura 9. Fragmento del archivo pom.xml

2.8 NGSIv2 Specification

NGSI [6] es una especificación propuesta por la OMA (Open Mobile Alliance) que se basa en una API Rest para comunicarse mediante peticiones HTTP. NGSI son las siglas de Next Generation Service Interface, que traducido resulta "interfaz de servicio de última generación". Recordar el concepto de información de contexto mencionado en la introducción, "información relevante sobre lo que está pasando en un momento dado, procedente de diferentes fuentes". NGSI está pensada para controlar el ciclo de vida completo de la información de contexto, incluyendo actualizaciones, peticiones, registros y suscripciones. Esta API define lo siguiente:

- Un modelo de datos para la información de contexto, basado en el concepto de entidades de contexto.
- Una interfaz de datos de contexto para el intercambio de información por medio de peticiones, suscripciones y actualizaciones.
- Una interfaz de disponibilidad del contexto, para intercambiar información de cómo obtener la propia información de contexto.

2.8.1 Modelado de la información de contexto

Los elementos principales en el modelo de datos de NGSI son las entidades de contexto, los atributos y los metadatos, como se puede observar en la figura de abajo.

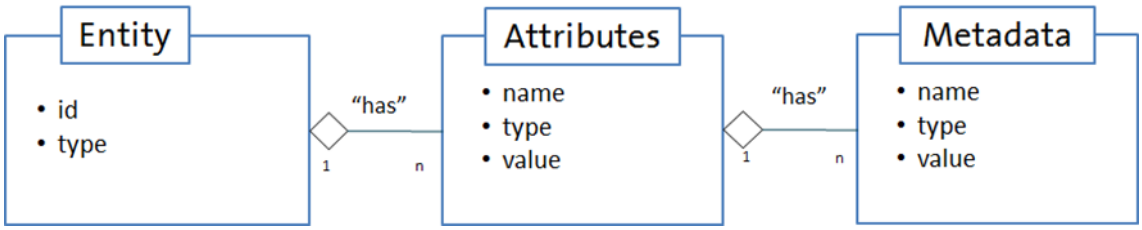

Figura 10. Componentes de una entidad de contexto

2.8.1.1 Entidades de contexto

Las entidades de contexto, o simplemente las entidades, son el centro de gravedad en el modelo de información FIWARE NGSI. Una entidad representa una cosa, por ejemplo, cualquier objeto físico o lógico. Puede ser un sensor, una persona, una habitación, etc. Cada entidad posee un identificador de entidad único.

Además, el sistema de tipos de FIWARE NGSI permite a las entidades tener un tipo de entidad. Los tipos de entidad están pensados para describir el tipo del objeto representado por la entidad. Por ejemplo, una entidad de contexto con el identificador "sensor-2000" podría tener el tipo "sensorTemperatura".

Cada entidad se identifica de forma unívoca mediante la combinación de su identificador (id) y su tipo (type).

2.8.1.2 Atributos de contexto

Los atributos de contexto son propiedades de las entidades de contexto. Por ejemplo, la velocidad de un coche se podría modelar como el atributo "velocidad_actual" de la entidad "coche-100".

En el modelo de datos NGSI, los atributos tienen un nombre, un tipo, un valor y metadatos asociados. El nombre del atributo describe qué tipo de propiedad representa el valor del atributo, como "velocidad_actual" en el caso del coche. El tipo del atributo representa el tipo del valor

NGSI del valor del atributo. Finalmente, se encuentran el valor del atributo en sí y una serie de metadatos opcionales que facilitan información sobre propiedades como el proveedor de la información, su precisión o su marca de tiempo entre otros.

2.8.1.3 Metadatos de contexto

Los metadatos de contexto se usan en varios lugares de FIWARE NGSI, siendo uno de ellos el caso descrito arriba. Es decir, una parte opcional que proporciona información extra del atributo. La estructura de los metadatos es la siguiente.

- Un nombre, que describe el papel del metadato en el lugar que ocurre.
- Un tipo, describiendo el tipo del valor NGSI del valor del metadato.
- Un valor, que contiene el metadato en sí.

2.8.2 Representación JSON de la información

Una entidad se representa mediante un objeto JSON con la siguiente sintaxis:

- El identificador de las entidades se especifica mediante la propiedad "id" del objeto, se modela como un string.

- El tipo de la entidad se especifica por la propiedad "type", también modelado como string.

- Los atributos de la entidad se especifican mediante propiedades adicionales, sus nombres son los del atributo que representan y sus posibles representaciones se discutirán en el siguiente apartado. Obviamente, los nombres id y type están reservados y no podrán ser usados como nombres de atributos.

Un ejemplo de esta sintaxis sería:

```
{
  "id": "entityID",
  "type": "entityType",
  "attr_1": <val_1>,
  "attr_2": <val_2>,
  ...
  "attr_N": <val_N>
}
```

Figura 11. Entidad NGSI en JSON

Por lo tanto, la representación normalizada de las entidades siempre incluye el campo id y type. Además de un número variable de atributos de la entidad. Existen otros tipos de representaciones, como son la simplificada o la parcial, que veremos más adelante. La especificación de cada operación incluye detalles sobre qué representación se espera como entrada, o qué representación se proporcionará a la salida.

2.8.2.1 Representación JSON de los atributos

Un atributo se presenta mediante un objeto JSON con la siguiente sintaxis:

- El valor del atributo se especifica mediante la propiedad "value", cuyo valor puede ser cualquier tipo de dato soportado por JSON.

- El tipo NGSI del atributo es especificado por la propiedad "type", cuyo valor es un string conteniendo el tipo.

- Los metadatos del atributo se especifican por la propiedad “metadata”. Su valor es otro objeto JSON que contiene una propiedad por cada metadato definido. El nombre de la propiedad es el nombre del metadato. A su vez, cada metadato se representa por otro objeto JSON que contiene lo siguiente:

    - value: el valor del metadato vale cualquier tipo soportado por JSON.
    - type: un string que contiene el tipo NGSI del metadato.

  Un ejemplo de atributo con varios metadatos sería:

```
{
  "value": <...>,
  "type": <...>,
  "metadata": {
      "meta_1": {
          "value": <...>,
          "type" : <...>
      },
      "meta_2": {
          "value": <...>,
          "type": <...>
      },
      ...
      "meta_N": {
          "value": <...>,
          "type": <...>
      }
  }

}
```

Figura 12. Atributo NGSI en JSON

2.8.2.2 Representación simplificada de la entidad

Hay dos modos de representación que deben ser soportadas por las implementaciones de NGSI.

- **Modo “keyValues”.** Este modo representa los atributos de la entidad en parejas nombre-valor, ignorando información sobre el tipo y los metadatos. También incluye información sobre el identificador y tipo de la entidad. El siguiente ejemplo describe una entidad tipo “Car” con identificador “car100” que tiene 3 atributos.

```
{
    "id": "car100",
    "type": "Car",
    "branch": "Ford",
    "color": "black",
    "engine:": 78.3

}
```

Figura 13. Representación tipo "keyValues"

- **Modo “values”.** Este modo representa una entidad como una lista de valores de atributos. Se deja fuera la información sobre el identificador, el nombre y el tipo. El orden de los atributos de la lista se especifica mediante el parámetro “attrs” en la URL de la petición HTTP. Por ejemplo, “attrs=branch,color,engine”. Si no se usa “attrs”, el orden es arbitrario.

```
[ "Ford", "black", 78.3]
```

Figura 14. Representación tipo "values"

2.8.2.3 Representación parcial de la entidad

Algunas operaciones usan la representación parcial de las entidades, según estas reglas:

- En las operaciones de actualización, no se permiten las propiedades “id” y “type” ya que son inmutables.

- En algunos casos, no se muestran todos los atributos de la entidad. Por ejemplo, una petición donde se selecciona expresamente un subconjunto de los atributos de la entidad.

- El campo “value” de un atributo o metadato puede ser omitido en la petición, lo que significa que toma el valor null. En las respuestas este valor siempre está presente.

- El campo “type” puede ser omitido en las peticiones. Cuando se omite en la creación de atributos o metadatos, o en operaciones de actualización, un valor por defecto es usado según el campo “value”:

    - Si es un string, se usa el tipo “Text”.
    - Si es un número, se usa el tipo “Number”.
    - Si es un boolean, se usa el tipo “Boolean”.
    - Si es un objeto o lista, se usa el tipo “StructuredValue”.
    - Si es null, se usa “None”.

- Los metadatos del atributo pueden ser omitidos en las peticiones, lo que significa que no hay elementos de metadatos asociados al atributo. En las respuestas, esta propiedad se pone a “{}” si el atributo no tiene ningún metadato asociado.

2.8.2.4 Tipos especiales de atributos

En general, los atributos definidos por el usuario son informativos y el servidor NGSIv2 los procesa de forma opaca. Esto quiere decir que no tienen ningún sentido especial para el servidor NGSIv2. No obstante, los tipos descritos a continuación sí tienen un sentido especial:

- “DateTime”: identifica fechas en el formato ISO8601. Por ejemplo:

```
{
  "timestamp": {
    "value": "2017-06-17T07:21:24.238Z",
    "type: "DateTime"
  }
}
```

Figura 15. Atributo tipo "DateTime"

- “geo:point”, “geo:line”, “geo:box”, “geo:polygon” y “geo:json”.Tiene una semántica especial relacionada con la posición física de la entidad.

2.8.3 Restricciones sintácticas

Los campos que se usan como identificadores en la API NGSIv2 siguen unas reglas especiales en lo que respecta a su sintaxis. Las reglas aplican a los campos:

- Identificador de entidad.
- Tipo de entidad.
- Nombre de atributo.
- Tipo de atributo.

- Nombre de metadato.
- Tipo de metadato.

Las reglas son las siguientes:

- Los caracteres permitidos son los que están en el conjunto ASCII standard, excepto los siguientes: caracteres de control, espacio en blanco, “&”, “?”, “/” y “#”.
- La longitud máxima del campo es de 256 caracteres.
- La longitud mínima del campo es de 1 carácter.

Además de las reglas mencionadas, algunas implementaciones de servidores NGSIv2 podrían añadir restricciones sintácticas adicionales en estos u otros campos para, por ejemplo, evitar ataques de inyección de código. En el caso de que el cliente intente usar un campo inválido, desde el punto de vista sintáctico, el cliente obtiene un mensaje HTTP 400 Bad Request con un mensaje explicando la causa del error.

2.8.3.1 Restricciones en los nombres de los atributos

Las siguientes palabras no puede ser usadas como nombre de atributos:

- “id”, ya que entraría en conflicto con el campo usado para representar el identificador de la entidad.

- “type”, ya que entraría en conflicto con el campo usado para representar el tipo de la entidad.

- “geo:distance”, entraría en conflicto con el string usado en “orderBy” para la proximidad al punto central.

- “dateCreated”, “dateModified” y “dateExpires” que son atributos especificados por el servidor y no pueden ser modificados directamente por el cliente.

- “*”, ya que tiene un significado especial. Se refiere a todos los atributos en general.

2.8.3.2 Restricciones en los nombres de los metadatos

Los siguientes string no deben ser usados como nombres de metadatos:

- “dateCreated”, “dateModified”, ”previousValue” y “actionType”, que son atributos especificados por el servidor y no pueden ser modificados directamente por el cliente.

- “*”, ya que tiene un significado especial. Se refiere a todos los metadatos en general.

2.8.4 Propiedades geoespaciales de las entidades

Las propiedades geoespaciales de una entidad de contexto pueden ser representadas mediante atributos de contexto regulares. La existencia de propiedades geoespaciales permite que existan peticiones en base a la localización geográfica.

Las implementaciones de NGSI deben soportar dos sintaxis diferentes:

- “Simple Location Format”. Está pensado como un formato muy ligero para los desarrolladores y usuarios, que permite añadir localización de forma rápida y fácil a sus entidades existentes.

- “GeoJSON”. GeoJSON es un formato de intercambio de datos geoespaciales basado en JSON. Proporciona una mayor flexibilidad, permitiendo la representación de la altitud de un punto, así como formas geoespaciales más complejas.

Las aplicaciones cliente son responsables de definir qué atributos representan propiedades geoespaciales. Típicamente, es un atributo llamado “location”, pero nada prohíbe casos de uso donde una entidad contiene más de un atributo geoespacial. Por ejemplo, una serie de localizaciones que se especifican en diferentes grados de granularidad, o mediante métodos de localización diferentes con distinta precisión.

2.8.4.1 Simple Location Format

Simple Location Format soporta geometrías básicas (un punto, una línea, una caja, un polígono) y cubre los casos de uso típicos a la hora de codificar localizaciones geográficas. Se inspira en GeoRSS Simple [23].

Destacar que Simple Location Format no está pensado para representar posiciones complejas en la superficie de la tierra. Por ejemplo, las aplicaciones que requieran capturar la altitud de las coordenadas deberán usar la representación GeoJSON.

Un atributo de contexto que represente una posición codificada según Simple Location Format debe presentar la siguiente sintaxis:

- El tipo del atributo debe ser uno de los siguientes: “geo:point”, “geo:line”,”geo:box” o “geo:polygon”.

- El valor del atributo debe ser una lista de coordenadas. Por defecto, las coordenadas se definen usando el sistema de referencia de coordenadas WGS84 Lat Long, EPSG::4326, con la latitud y la longitud en grados decimales. Tal lista de coordenadas permite codificar la geometría especificada por el campo “type” y se codifican según las reglas específicas definidas a continuación:

    - Tipo “geo:point”: el valor del atributo debe contener un string con un par válido de latitud-longitud, separados por una coma.
    - Tipo “geo:line”: el valor del atributo debe contener una lista de string de pares válidos latitud-longitud. Al menos debe haber dos pares.
    - Tipo “geo:polygon”: el valor del atributo debe contener una lista de string de pares válidos latitud-longitud. Debe haber al menos 4 pares, siendo el último idéntico al primero, por lo tanto un polígono tiene al menos 3 puntos distintos. Los pares de coordenadas deben ser ordenados correctamente para que los segmentos de línea que forman el polígono permanezcan en el borde exterior del área definida.
    - Tipo “geo:box”: una caja limitante no es más que una región rectangular, a menudo usada para definir el alcance de un mapa o un área aproximada de interés. Se representa mediante una lista de pares latitud-longitud de tamaño 2, modelado como string. El primer par indica la esquina inferior y el segundo la superior.

- Como se ve en los dos ejemplos siguientes, un punto geográfico se define mediante su latitud y longitud. Una caja se define mediante dos puntos, el primero indica la esquina inferior y el segundo la superior.

```
{
  "location": {
    "value": "41.3763726, 2.186447514",
    "type": "geo:point"
  }
}
```

```
{
  "location": {
    "value": [
      "40.63913831188419,
-8.653321266174316",
      "40.63881265804603,
-8.653149604797363"
    ],
    "type": "geo:box"
  }
}
```

Figura 16. Localización según Simple Location Format

2.8.4.2 GeoJSON

Un atributo de contexto que represente una localización según GeoJSON debe seguir la siguiente sintaxis:

- El tipo NGSI del atributo debe ser "geo:json".
- El valor del atributo debe ser un objeto GeoJSON válido. Destacar que la longitud se pone antes que la latitud en las coordenadas GeoJSON.

A continuación, se ilustra un ejemplo del uso de GeoJSON. Para más información recurrir a esta página web [24].

```
{
  "location": {
    "value": {
      "type": "Point",
      "coordinates": [2.186447514,
41.3763726]
    },
    "type": "geo:json"
  }
}
```

Figura 17. Localización según GeoJSON

2.8.5 Mensajes de Notificación

Los mensajes de notificación tienen dos campos:

- "subscriptionId": representa la suscripción que originó la notificación.

- "data": es una lista con los datos de la notificación en sí, incluye la entidad y todos los atributos de interés. Cada elemento de la lista corresponde a una entidad diferente. Por defecto, las entidades se representan en el modo normalizado. Sin embargo, mediante el modificador "attrsFormat", se puede pedir la representación simplificada.

En el campo "notification" del objeto Subscription, descrito en el siguiente apartado, se encuentra el parámetro "attrsFormat". Este parámetro indica el modo de representación de los datos contenidos en el mensaje de notificación. Si este parámetro es igual a "normalized" o si directamente se omite, la representación usada es la siguiente:


```
{
  "subscriptionId": "12345",
  "data": [
    {
      "id": "Room1",
      "type": "Room",
      "temperature": {
        "value": 23,
        "type": "Number",
        "metadata": {}
      },
      "humidity": {
        "value": 70,
        "type": "percentage",
        "metadata": {}
      }
    },
    {
      "id": "Room2",
      "type": "Room",
      "temperature": {
        "value": 24,
        "type": "Number",
        "metadata": {}
      }
    }
  ]
}
```


Figura 18. Notificación NGSI por defecto

Si el parámetro “attrsFormat” es igual a “keyValues”, entonces se usa el modo de representación simplificada:


```
{
  "subscriptionId": "12345",
  "data": [
    {
      "id": "Room1",
      "type": "Room",
      "temperature": 23,
      "humidity": 70
    },
    {
      "id": "Room2",
      "type": "Room",
      "temperature": 24
    }
  ]
}
```


Figura 19. Notificación NGSI con representación "keyValues"

Si el parámetro “attrsFormat” es igual a “values”, entonces se usa el modo de representación parcial:


```
{
  "subscriptionId": "12345",
  "data": [ [23, 70], [24] ]
}
```


Figura 20. Notificación NGSI con representación “values”

Es obligatorio usar la cabecera HTTP “Ngsiv2-AttrsFormat” en los mensajes de notificación con el valor del formato de la suscripción asociada. Así, los que reciban la notificación son conscientes del formato sin necesidad de procesar el cuerpo del mensaje.

2.8.6 Suscripciones

Una suscripción en FIWARE se representa mediante un objeto JSON **subscription** con los siguientes campos:

- “id”: identificador único de la suscripción. Es creado automáticamente cuando se crea el objeto Subscription en el Context Broker.
- “description”: es un campo de texto libre usado por el cliente para describir la suscripción. Este campo es opcional.
- **“subject”:** es un objeto que describe qué entidades se verán afectadas por la suscripción.
- **“notification”:** es un objeto que describe la notificación que debe enviar el sistema cuando salte la suscripción.
- “expires”: indica la fecha en la que caduca la suscripción en formato ISO8601. En caso de no aparecer este campo, la suscripción sería permanente.
- “status”: indica el estado de la suscripción. Puede ser **active** (para las suscripciones activas) o **inactive** (para las suscripciones inactivas). En caso de no especificar este campo a la hora de la creación de la suscripción, por defecto las nuevas suscripciones se crean con este campo igual a **active**. Esto puede ser cambiado a posteriori por el cliente. En cuanto a las suscripciones caducadas, este campo toma el valor de **expired** y no puede ser cambiado por el cliente. Además, en caso de que la suscripción sufra problemas a la hora de notificar, este campo toma el valor de **failed**. En cuanto se solucione el problema vuelve a cambiarse a **active**.
- “throttling”: indica el periodo mínimo de tiempo en segundos que debe pasar entre dos notificaciones consecutivas. Este campo es opcional.

Un objeto “**subject**” contiene los siguientes subcampos:

- “entities”: es una lista de objetos, cada uno de ellos con los siguientes subcampos. Aclarar que las posibilidades de los patrones son muy extensas, no se van a especificar en detalle porque solo se ha usado un tipo muy concreto que es el patrón “^prueba”, para filtrar los identificadores de las entidades por su comienzo:
    - “id” or “idPattern”: identificador o patrón de las entidades afectadas. No se pueden usar ambos al mismo tiempo, pero al menos uno de los dos tiene que estar presente obligatoriamente. Un ejemplo de patrón sería “.*” para seleccionar todos los identificadores del servidor.
    - “type” or “typePattern”: tipo o patrón aplicado al tipo de las entidades afectadas. No se pueden usar ambos a la vez. En caso de no aparecer ninguno, significa que cualquier tipo es válido.
- “condition”: condición que desencadena la notificación. Este campo es opcional y puede contener además otras dos propiedades opcionales también.

- “attrs”: lista de nombres de atributos. Esto implica que cualquier cambio en los atributos indicados en esta lista, dentro de las entidades seleccionadas por el campo entities, desencadenará una notificación.

- “expression”: una expresión compuesta por alguna o algunas de estas condiciones: “q”, “mq”, “georel”, “geometry” y “coords”. Estas condiciones sirven para realizar filtros.

    - “q”: es utilizada para filtrar valores de atributos.
    - “mq”: es utilizada para filtrar valores de metadatos.
    - “georel”: es utilizada para especificar una relación espacial.
    - “geometry”: es utilizada para definir una forma de referencia usada para “resolver” la petición (point,line,polygon,box).
    - “coords: debe ser un string, que contenga una lista de pares de coordenadas geográficas separadas por “;”. Las coordenadas tienen que concordar con la geometría indicada en geometry.

Un objeto **notification** tiene los siguientes subcampos:

- “attrs” or “exceptAttrs”: no pueden estar ambos presentes al mismo tiempo

    - “attrs”: lista de atributos que deben ser incluidos en el mensaje de notificación. Una lista vacía significa que todos los atributos deben ser incluidos.

    - “exceptAttrs”: lista de atributos que deben ser excluidos del mensaje de notificación. Por lo tanto, el mensaje de notificación incluye todos los atributos menos los indicados en esta lista.

- **“http” or “httpCustom”:** uno de los dos tiene que estar presente, pero no ambos al mismo tiempo. Este campo se usa para especificar parámetros del protocolo HTTP que deben ser aplicados en el envío de las notificaciones.

- “attrsFormat” (opcional): especifica cómo deben ser representadas las entidades dentro de las notificaciones. Los valores acepteptados son:
    - normalized
    - keyValues
    - values
    - En caso de que el campo attrsFormat tome un valor diferente a los anteriores el sistema Orion dará un error en la suscripción.

- “metadata” (opcional): lista de los metadatos que deben ser incluidos en los mensajes de notificación.

- “timesSent”: este campo no es editable, solo está presente cuando hacemos un GET a la suscripción. Representa el número de notificaciones mandadas como resultado de esta suscripción.

- “lastNotification”: este campo no es editable, solo está presente cuando hacemos un GET a la suscripción. Representa el instante en el que se realizó la última notificación en formato ISO8601.

- “lastFailure”: este campo no es editable, solo está presente cuando hacemos un GET a la suscripción. Representa el instante en el que tuvo lugar el último fallo en el formato ISO8601. Este campo no está presente en el caso de que nunca haya habido un problema con las notificaciones.

- “lastSuccess”: Este campo no es editable, solo está presente cuando hacemos un GET a la suscripción. El instante de tiempo, según el formato ISO8601, en el que se realizó la última notificación con éxito. Este campo no está presente si la suscripción nunca ha tenido una notificación exitosa.

Un objeto “**http”** tiene los siguientes subcampos:

- “url”: URL que referencia al servicio que debe ser invocado cuando se genera una notificación. Un servidor que cumpla NGSIv2, deber soportar el esquema http. Otros esquemas también son posibles.

Un objeto “**httpCustom”** tiene los siguientes subcampos:

- “url”: igual que en el campo de arriba.

- “headers”: este campo es opcional. Mapa de las cabeceras HTTP que deben ser incluidas en los mensajes de notificación.

- “qs”: este campo es opcional. Mapa de los parámetros de la petición URL que deben ser incluidos en los mensajes de notificación.

- “method”: este campo es opcional. El método HTTP a usar cuando se envía la notificación. Por defecto es POST. Sólo son válidos los métodos HTTP. En caso de especificar un método no válido, saltaría el error 400 Bad Request.

- “payload”: este campo es opcional. El payload que debe ser usado en las notificaciones. En caso de ser omitido, se usa el payload por defecto.

2.8.7 Respuestas en caso de error

En el cuerpo del mensaje de error se encuentra un objeto JSON con los siguientes campos:

- **error**: es un campo obligatorio, modelado por un string. Consiste en una descripción textual del error.

- **description**: es un campo opcional, modelado por un string. Ofrece información adicional sobre el error.

Todas las implementaciones de un servidor NGSIv2 están obligadas a usar los siguientes códigos de estado HTTP y los siguientes textos en el campo **error**. No obstante, el texto usado en el campo **description** depende de la implementación.

- En caso de que el objeto JSON en el cuerpo de la petición no pueda ser procesado se devuelve ParseError (400).

- Los errores causados únicamente por la petición (no dependen del estado del servidor), ya sea en los parámetros de la URL o en el cuerpo, devuelven BadRequest (400).
    - Excepción: el primer caso es una excepción de este caso.

- El intento de exceder el límite del índice espacial devuelve NoResourceAvailable (413).

- En caso de ambigüedad porque la petición puede referirse a más de un recurso, se devuelve TooManyResults (409). Esto puede ocurrir si no se indica el tipo de la entidad.

- Si el recurso identificado por la petición no se encuentra, se devuelve NotFound (404).
- En caso de errores debidos al tipo de petición combinados con el estado del recurso. Por ejemplo, cuando se intenta realizar un POST con el parámetro options=append en un atributo ya existente. Esto resulta en el error Unprocessable (422).
    - Excepciones: Los casos anteriores que devuelven 404, 409 o 413.
- Los errores debidos a la capa HTTP son los siguientes:
    - HTTP 405 Method Not Allowed, se corresponde con MethodNotAllowed (405).
    - HTTP 411 Length Required, se corresponde con ContentLengthRequired (411).
    - HTTP 413 Request Entity Too Large, se corresponde con RequestEntityTooLarge (413).
    - HTTP 415 Unsupported Media Type, se corresponde con UnsupportedMediaType (415).

2.8.8 Puntos de entrada de la API NGSIv2

2.8.8.1 Crear una entidad

Figura 21. Petición POST para crear una entidad

Para crear una entidad se envía un mensaje HTTP POST. La dirección de destino se forma cogiendo la URL del servidor NGSIv2 y añadiéndole "/v2/entities". Además, existe un parámetro opcional llamado "options" cuyos valores posibles son "keyValues" y "upsert".

Cuando se usa "keyValues", el cuerpo de la petición usa la representación simplificada "keyValues" explicada con anterioridad. Cuando se usa "upsert", la entidad se actualiza si ya existe en el sistema. Si no se usa "upsert" y la entidad ya existe en el sistema se devuelve el error 442 Unprocessable Entity.

Destacar que el mensaje debe usar la cabecera HTTP "Content-Type:application/json" y que el cuerpo del mensaje es un objeto JSON que represente a la entidad que debe ser creada.

Las respuestas posibles son las siguientes:

- Las operaciones con éxito usan el código de respuesta 201, si no se usa la opción "upsert", o el código 204 en el caso de que se use "upsert". El mensaje de respuesta incluye una cabecera llamada "Location" con la URL de la entidad creada.
- Ver la sección anterior "Respuestas en caso de error".

2.8.8.2 Recuperar una entidad por su id

Figura 22. Petición GET para recuperar una entidad

Para recuperar una entidad por su id, se envía un mensaje HTTP GET. La dirección de destino se forma cogiendo la URL del servidor NGSIv2 y añadiéndole "v2/entities/{identificador de la entidad}". Encontramos los siguientes parámetros:

- type: es el tipo de la entidad, sirve para evitar ambigüedades en el caso de que varias entidades de diferente tipo tengan el mismo identificador.
- attrs: lista de los atributos de la entidad solicitados, son los únicos atributos que se devuelven. En el caso de omitirse este campo, se incluyen en la respuesta todos los atributos de la entidad en un orden arbitrario.
- metadata: una lista de nombres de metadatos que deben ser incluidos en la respuesta.
- options: los posibles valores de este campo son "keyValues" y "values". Permite elegir algún modo especial de representar los datos en la respuesta.

La respuesta a esta petición es un objeto en formato JSON que representa a la entidad identificada por el campo id. Esta operación solo debe devolver un único elemento entidad.

En el caso en el que haya varias entidades asociadas al mismo identificador, se devuelve un mensaje de error, con el código de estado HTTP 409, indicando que ha habido un conflicto. Esto ocurre cuando no se indica el tipo de la entidad en el campo type.

Las operaciones con éxito devuelven el código 200 OK. El resto de los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error".

2.8.8.3 Recuperar atributos de una entidad

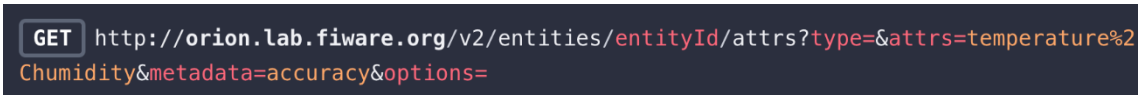

Figura 23. Petición GET para recuperar atributos de una entidad

Esta petición es muy parecida a la anterior. La primera diferencia es que a la URL se le añade "/attrs" para indicar que en la respuesta deben omitirse los campos "id" y "type". La segunda es obviamente que en la respuesta solo se encuentran los atributos de la entidad y no la entidad completa como en el caso de arriba. El resto es todo igual.

2.8.8.4 Actualizar o añadir atributos a una entidad

Figura 24. Petición POST para actualizar/añadir atributos

Para actualizar o añadir atributos a una entidad enviamos un mensaje HTTP POST. Hay que recalcar que la elección del método POST la establece el estándar NGSIv2, ya que se puede pensar que sería más correcto utilizar el método PUT. La dirección de destino se forma cogiendo como base la URL del servidor NGSIv2 y añadiéndole "v2/entities/{identificador de la entidad}". Los parámetros son:

- type: es el tipo de la entidad, sirve para evitar ambigüedades en el caso de que varias entidades de diferente tipo tengan el mismo identificador.
- options: los posibles valores de este campo son "keyValues" y "append".

El cuerpo de la petición consiste en un objeto JSON representado los atributos que añadir o actualizar. Si no se usa "append", los atributos de la entidad se actualizan (si existían previamente) o se añaden (si no existían con anterioridad) con los encontrados en el cuerpo del mensaje. En el caso de usar "append", todos los atributos del cuerpo del mensaje que no existieran previamente son añadidos a la entidad. Además, si hubiera algún atributo del mensaje que ya existiera en la entidad, se devolvería un mensaje de error.

Las operaciones con éxito usan el código 204. Los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error".

2.8.8.5 Actualizar atributos ya existentes de una entidad

```
PATCH http://orion.lab.fiware.org/v2/entities/entityId/attrs?type=&options=
```

Figura 25. Petición PATCH para actualizar atributos

Para actualizar atributos ya existentes de una entidad, enviamos un mensaje HTTP PATCH. La dirección de destino la conseguimos añadiéndole a la URL base del servidor NGSIv2 "/v2/entities/{identificador de la entidad}/attrs". Tenemos dos parámetros:

- type: es el tipo de la entidad, sirve para evitar ambigüedades en el caso de que varias entidades de diferente tipo tengan el mismo identificador.
- options: el único valor posible de este campo es "keyValues".

El cuerpo de la petición es un objeto en formato JSON representando los atributos para actualizar. Es decir, consiste en una entidad, pero sin los campos id y type. Estos campos se prohíben en esta petición. Los atributos de la entidad se actualizan con los del cuerpo del mensaje. En el caso de que uno o más atributos presentes en el mensaje no existan en la entidad, se devuelve un mensaje de error. Hay que aclarar que los atributos ya existentes en la entidad que no son mencionados en esta operación se mantienen en la entidad. Esta es la diferencia con la siguiente petición donde se utiliza PUT en lugar de PATCH.

Las operaciones con éxito usan el código 204. Los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error".

2.8.8.6 Sustituir todos los atributos de una entidad

```
PUT http://orion.lab.fiware.org/v2/entities/entityId/attrs?type=&options=
```

Figura 26. Petición PUT para sustituir atributos

Para sustituir todos los atributos de una entidad, enviamos un mensaje HTTP PUT. La dirección de destino la conseguimos añadiéndole a la URL base del servidor NGSIv2 "/v2/entities/{identificador de la entidad}/attrs". Tenemos dos parámetros:

- type: es el tipo de la entidad, sirve para evitar ambigüedades en el caso de que varias entidades de diferente tipo tengan el mismo identificador.
- options: el único valor posible de este campo es "keyValues".

El cuerpo de la petición es un objeto en formato JSON representando los atributos para actualizar. No se permiten los campos "id" ni "type". Los atributos previamente existentes de la entidad se eliminan y son sustituidos por los del cuerpo del mensaje. Es decir, cualquier atributo que existía antes es eliminado si no aparecen en el objeto JSON de la nueva solicitud.

Las operaciones con éxito usan el código 204. Los códigos y mensajes de error están explicados la sección anterior “Respuestas en caso de error”.

2.8.8.7 Eliminar una entidad

Figura 27. Petición DELETE para eliminar una entidad

Para eliminar una entidad enviamos un mensaje HTTP DELETE. La URI destino se forma cogiendo la URL base del servidor NGSIv2 y añadiendo “/v2/entities/{identificador de la entidad}”. Tenemos un único parámetro adicional para indicar el tipo de la entidad a eliminar.

En caso de éxito, la respuesta es un mensaje HTTP con el código 204. Los códigos y mensajes de error están explicados la sección anterior “Respuestas en caso de error”. Como recordatorio, si no se especifica el tipo de la entidad y hay ambigüedad en la petición, se devuelve un mensaje de error.

2.8.8.8 Recuperar un atributo concreto de una entidad

Figura 28. Petición GET para recuperar un atributo concreto

Para recuperar los datos de un atributo concreto dentro de una entidad, enviamos un mensaje HTTP GET. La URL destino se obtiene con la dirección base del servidor NGSIv2, más “v2/entities/{id de la entidad}/attrs/{nombre del atributo}”. Encontramos los siguientes parámetros:

- type: es el tipo de la entidad, sirve para evitar ambigüedades en el caso de que varias entidades de diferente tipo tengan el mismo identificador.
- attrName: nombre del atributo a recuperar
- metadata: una lista de nombres de metadatos que deben ser incluidos en la respuesta.

Esta petición devuelva un objeto JSON con los datos del atributo. En caso de una operación exitosa se devuelve un mensaje con el código 200 OK. Los códigos y mensajes de error están explicados la sección anterior “Respuestas en caso de error”. En el apartado 2.8.8.3 “Recuperar atributos de una entidad”, se pueden recuperar o todos los atributos de la entidad o una selección de ellos. En este caso sólo se recupera un único atributo. La diferencia con el apartado 2.8.8.11 “Recuperar el valor de un atributo concreto de una entidad” consiste en que en ese apartado solo se obtiene el valor del campo “value” dentro del atributo. Mientras que aquí se obtienen todos los otros campos del atributo también.

2.8.8.9 Actualizar un atributo concreto de una entidad

Figura 29. Petición PUT para actualizar un atributo concreto

Para actualizar los datos de un atributo dentro de una entidad, enviamos un mensaje HTTP PUT. La URL destino se obtiene con la dirección base del servidor NGSIv2, más “v2/entities/{id de la entidad}/attrs/{nombre del atributo}”. Encontramos los siguientes parámetros:

- type: es el tipo de la entidad, sirve para evitar ambigüedades en el caso de que varias entidades de diferente tipo tengan el mismo identificador.
- attrName: nombre del atributo a actualizar.

El cuerpo de la petición es un objeto en formato JSON representando los nuevos datos del atributo. Los datos anteriores son eliminados y sustituidos por los nuevos del cuerpo del mensaje. En caso de operación con éxito, la respuesta es un mensaje con el código 204. Los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error".

2.8.8.10 Eliminar un atributo concreto de una entidad

```
DELETE http://orion.lab.fiware.org/v2/entities/entityId/attrs/attrName?type=
```

Figura 30. Petición DELETE para eliminar un atributo concreto

Para eliminar un solo atributo de una entidad, enviamos un mensaje HTTP DELETE. La URL destino la conseguimos con la dirección base del servidor NGSIv2 y añadiéndole "/v2/entities/{identificador de la entidad}/attrs/{nombre del atributo}". Encontramos los siguientes parámetros:

- type: es el tipo de la entidad, sirve para evitar ambigüedades en el caso de que varias entidades de diferente tipo tengan el mismo identificador.
- attrName: nombre del atributo a eliminar.

En caso de que se haya borrado con éxito, recibimos un mensaje con el código 204. Los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error".

2.8.8.11 Recuperar el valor de un atributo concreto de una entidad

```
GET http://orion.lab.fiware.org/v2/entities/entityId/attrs/attrName/value?type=
```

Figura 31. Petición GET para recuperar el valor de un atributo

Para recuperar únicamente el valor de un atributo concreto, enviamos un mensaje HTTP GET. La URL destino se forma con la URL base del servidor NGSIv2 y añadiéndole "/v2/entities/{identificador de la entidad}/attrs/{nombre del atributo}/value". Encontramos los siguientes parámetros:

- type: es el tipo de la entidad, sirve para evitar ambigüedades en el caso de que varias entidades de diferente tipo tengan el mismo identificador.
- attrName: nombre del atributo a recuperar.

Esta operación devuelve el valor del atributo seleccionado:

- Si el valor es un objeto o lista JSON:
    - Si la cabecera HTTP "Accept" puede ser expandida a "application/json" o "text/plain" se devuelve el valor en formato JSON con una respuesta de tipo "application/json" o "text/plain". Se elige la que esté primero en la cabecera "Accept" o "application/json" en el caso "Accept: */*".
    - Si no se cumple lo de arriba se devuelve el error HTTP : "406 Not Acceptable: accepted MIME types: application/json, text/plain"-
- Si el valor es un string, numero, null o booleano:
    - Si la cabecera HTTP "Accept" se puede expandir a "text/plain" se devuelve el valor como text plano. En el caso de ser un string, se usan comillas dobles al inicio y al final.
    - Si no se cumple lo de arriba se devuelve el error HTTP: "406 Not Acceptable: accepted MIME types: text/plain".

Las operaciones con éxito usan el código 200 OK. Los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error".

2.8.8.12 Actualizar el valor de un atributo concreto de una entidad

Figura 32. Petición PUT para actualizar el valor de un atributo

Para actualizar únicamente el valor (campo "value") de un atributo concreto, enviamos un mensaje HTTP PUT. La URL destino se forma con la URL base del servidor NGSIv2 y añadiéndole "/v2/entities/{identificador de la entidad}/attrs/{nombre del atributo}/value". Encontramos los siguientes parámetros:

- type: es el tipo de la entidad, sirve para evitar ambigüedades en el caso de que varias entidades de diferente tipo tengan el mismo identificador.
- attrName: nombre del atributo a actualizar.

En el cuerpo de la petición encontramos el nuevo valor para el atributo.

- Si el tipo MIME del cuerpo de la petición es "application/json", entonces el valor del atributo se iguala al objeto o lista JSON codificada en el cuerpo. Si el JSON no es válido se devuelve un mensaje de error.

- Si el tipo MIME del cuerpo de la petición es "text/plain", entonces se aplica el siguiente algoritmo:

    - Si el cuerpo empieza y termina con comillas dobles, el valor se considera como un string. Hay que aclarar que las comillas no son parte del string.
    - Si es "true" o "false", el valor se toma como un booleano.
    - Si es "null", se toma el valor null.
    - En caso de que estos 3 primeros tests fallen, se interpreta como un número.
    - En caso de no ser un número válido, se devuelve un mensaje de error y el atributo no se modifica.

El tipo MIME del cuerpo de la petición se indica en la cabecera HTTP "Content-Type". En caso de éxito se devuelve un mensaje con el código 204. Los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error". Realmente, aunque el objetivo de esta petición sea modificar el campo value del atributo, de forma indirecta posiblemente también se cambie su tipo NGSI como se indica, por ejemplo, en el caso del tipo MIME "text/plain".

2.8.8.13 Listar suscripciones

Figura 33. Petición GET para listar suscripciones

Para obtener una lista de las suscripciones existentes, enviamos un mensaje HTTP GET. La URL destino se forma con la URL base del servidor NGSIv2 y añadiéndole "/v2/subscriptions". Encontramos los siguientes parámetros:

- limit: sirve para limitar el número de suscripciones que deben ser recuperadas.
- offset: sirve para saltar un número de suscripciones.
- options: la única opción disponible es "count".

Se devuelve una lista JSON con todos los objetos suscripción presentes en el sistema. En caso de éxito se devuelve un mensaje con el código 200 OK. Los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error".

2.8.8.14 Crear suscripciones

Figura 34. Petición POST para crear una suscripción

Para crear una nueva suscripción enviamos un mensaje HTTP POST. La URL destino se forma con la URL base del servidor NGSIv2 y añadiéndole "/v2/subscriptions". No tenemos ningún parámetro adicional.

En el cuerpo de la petición encontramos la suscripción en formato JSON. En caso de éxito se devuelve un mensaje con el código 201 Created. Los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error".

2.8.8.15 Recuperar una suscripción según su ID

Figura 35. Petición GET para recuperar una suscripción

Para obtener una suscripción según su ID enviamos un mensaje HTTP GET. Las URL destino es la dirección base del servidor más "/v2/subscriptions/{identificador de la suscripción}".

La respuesta es la suscripción representada como un objeto JSON. En caso de éxito se devuelve un mensaje con el código 200 OK. Los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error".

2.8.8.16 Actualizar una suscripción

Figura 36. Petición PATCH para actualizar una suscripción

Para actualizar una suscripción usamos el mensaje HTTP PATCH. Las URL destino es la dirección base del servidor más "/v2/subscriptions/{identificador de la suscripción}".

En el cuerpo del mensaje se incluye los campos de la suscripción que deben ser actualizados. En caso de éxito se devuelve un mensaje con el código 204 No Content. Los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error". Aclarar que no podría usar PUT ya que el estándar NGSI obliga a usar PATCH.

2.8.8.17 Borrar una suscripción

Figura 37. Petición DELETE para borrar una suscripción

Para borrar una suscripción, usamos un mensaje HTTP DELETE. Las URL destino es la dirección base del servidor más "/v2/subscriptions/{identificador de la suscripción}".

En caso de éxito se devuelve un mensaje con el código 204 No Content. Los códigos y mensajes de error están explicados la sección anterior "Respuestas en caso de error".

2.9 FIWARE

2.9.1 Modelo de datos

La plataforma FIWARE alberga una serie de modelos de datos [25], enfocados principalmente al dominio de ciudades inteligentes. Podemos encontrar modelos para edificios, objetos de iluminación, plazas de aparcamiento y vehículos de transporte público entre muchos otros. Este estudio particular, se centra en dos modelos de datos concretos orientado a dispositivos. Tenemos al Device y el DeviceModel.

2.9.1.1 Device

Un recurso Device [26], se define como un dispositivo (esto incluye hardware, software y firmware) pensado para llevar a cabo una tarea concreta, puede ser actuar sobre algo, medir una variable del entorno, etc. Este recurso representa a un objeto tangible que contiene algo de lógica y produce y/o consume información. Se presupone que un Device, siempre es capaz de comunicar de forma electrónica por medio de alguna red.

A continuación, se expondrá como se define a nivel del modelo de datos el recurso Device:

Nombre	Obligatorio	Tipo	Descripción
id	X	-	Identificador único del dispositivo.
type	X	-	Tipo de la entidad. Debe ser igual a Device.
source		Property. Text o URL	Secuencia de caracteres que indica la fuente de los datos de la entidad.
dataProvider		Property. URL	URL a información sobre el proveedor que informó sobre esta entidad.
category		Lista de Text	Ver el atributo category del recurso DeviceModel descrito a continuación. Este atributo es opcional en este recurso Device, pero es recomendable incorporarlo para optimizar las búsquedas.
controlledProperty		Lista de Text	Ver el atributo controlledProperty del recurso DeviceModel descrito a continuación. Es opcional, pero se recomienda para optimizar las posteriores búsquedas.
controlledAsset		Lista de Text	Los recursos (edificios, objetos, etc) controlados por este dispositivo concreto.
mnc		Property. Text	Esta propiedad identifica el MNC, Mobile Network Code de la red a la que el dispositivo está conectada. El MNC se usa en combinación con el MCC (descrito justo abajo) para identificar de forma única un operador telefónico que use las redes GSM, CDMA, iDEN, TETRA y 3G/4G. Además de algunas redes satélite.
mcc		Property. Text	Corresponde a Mobile Country Code, esta propiedad identifica de forma unívoca el país al que pertenece la red móvil a la que está conectada el dispositivo.
macAddress		Lista de Text	La dirección MAC del dispositivo.
ipAddress		Lista de Text	La dirección IP del dispositivo. Puede una ser una lista de valores separados por comas, en el caso del que dispositivo posea más de una dirección IP.
supportedProtocol		Lista de Text	Ver el atributo supportedProtocol del recurso DeviceModel. Se necesita en el caso de que debido a una actualización software, se añada compatibilidad con nuevos protocolos. De otra forma, es mejor representarlo a nivel de recurso DeviceModel.

configuration		Property. StructuredValue	Las configuraciones técnicas del dispositivo. Este atributo está pensado para ser un diccionario de propiedades que describen parámetros que tienen que ver con la configuración de un dispositivo. Ya sean tiempos de expiración, periodos de informe de datos y otros parámetros que no estén cubiertos por el estándar actual. Metadatos del atributo: • **dateModified**: último instante en el que se modificó o Tipo: DateTime o Solo lectura. Se genera automáticamente.
location		GeoProperty. geo:json	Localización de este dispositivo representado como un punto geométrico en el formato GeoJSON. Normativa de referencia: https://tools.ietf.org/html/rfc7946
name		Property. Text	Nombre nemotécnico que se le da al dispositivo. Normativa de referencia: https://schema.org/name
description		Property. Text	Descripción del dispositivo. Normativa de referencia: https://schema.org/description
dateInstalled		Property. DateTime	Una marca de tiempo que indica cuándo se instaló el dispositivo.
dateFirstUsed		Property. DateTime	Una marca de tiempo que indica cuándo el dispositivo fue usado por primera vez.
dateManufactured		Property. DateTime	Una marca de tiempo que indica cuándo se fabricó el dispositivo
hardwareVersion		Property. Text	La versión del hardware del dispositivo
softwareVersion		Property. Text	La versión del software del dispositivo
firmwareVersion		Property. Text	La versión del firmware del dispositivo.
osVersion		Property. Text	La versión del sistema operativo del dispositivo.
dateLastCalibration		Property. DateTime	Una marca de tiempo que indica cuándo el dispositivo fue calibrado por última vez.
serialNumber		Property. Text	El número de serie asignado por el fabricante.
provider		Property. Provider	El proveedor del dispositivo.
refDeviceModel		Property. Ref(DeviceModel)	Referencia del recurso DeviceModel correspondiente a este dispositivo.
batteryLevel		Number	El nivel de batería del dispositivo. Cuando la batería está llena debe ser igual a 1.0 y 0.0 cuando está vacía. Si no se puede determinar, se indica con – 1. Metadatos del atributo: • **timestamp**: último instante en el que se actualizó o Tipo: DateTime o TimeInstant
rssi		Number	Indicador de la intensidad de la señal recibida para un dispositivo compatible con tecnologías inalámbricas.

			Deber ser igual a 1.0 cuando la intensidad de la señal sea máxima y 0.0 cuando no haya señal. Cuando no pueda ser determinada se indica con -1. Metadatos del atributo: • **timestamp**: último instante en el que se actualizó o Tipo: DateTime o TimeInstant
deviceState		Property. Text	Estado de este dispositivo desde el punto de vista operacional. Su valor puede depender del fabricante del dispositivo. Metadatos del atributo: • **timestamp**: último instante en el que se actualizó o Tipo: DateTime o TimeInstant
dateLastValueReported		Property. DateTime	Una marca de tiempo que indica cuándo fue la última vez que el dispositivo envió datos a la nube
value		Property. Text	Un valor observado o informado por el dispositivo. En el caso de actuadores, este atributo permite a un controlador cambiar la configuración de actuación. Por ejemplo, un dispositivo interruptor que esté encendido puede informar un valor de “on” de tipo Text en este campo. Obviamente, para cambiar de estado este interruptor, este atributo puede tomar el valor de “off”. Metadatos del atributo: • **timestamp**: último instante en el que se actualizó o Tipo: DateTime o TimeInstant
dateModified		Property. DateTime	Marca de tiempo que indica cuándo se modificó esta entidad por última vez. Sólo de lectura. Se genera automáticamente.
dateCreated		Property. DateTime	Marca de tiempo que indica cuándo se creó la entidad. Sólo de lectura. Se genera automáticamente.
owner		Property. Lista de URIs	Propietario del dispositivo.

Tabla 1. Recurso Device de FIWARE

```
{
    "id": "device-9845A",
    "type": "Device",
    "category": {
        "value": ["sensor"]
    },
    "batteryLevel": {
        "value": 0.75
    },
    "dateFirstUsed": {
        "type": "DateTime",
        "value": "2014-09-11T11:00:00Z"
    },
    "controlledAsset": {
        "value": ["wastecontainer-Osuna-100"]
    },
    "serialNumber": {
        "value": "9845A"
    },
    "mcc": {
        "value": "214"
    },
    "value": {
        "value": "l%3D0.22%3Bt%3D21.2"
    },
    "refDeviceModel": {
        "type": "Relationship",
        "value": "myDevice-wastecontainer-sensor-345"
    },
    "rssi": {
        "value": 0.86
    },
    "controlledProperty": {
        "value": ["fillingLevel", "temperature"]
    },
    "owner": {
        "value": ["http://person.org/leon"]
    },
    "mnc": {
        "value": "07"
    },
    "ipAddress": {
        "value": ["192.14.56.78"]
    },
    "deviceState": {
        "value": "ok"
    }
}
```


Figura 38. Recurso Device de FIWARE en JSON

2.9.1.2 DeviceModel

La entidad DeviceModel [27] captura las propiedades estáticas comunes a varias instancias de un Device.

A continuación, se expondrá como se define a nivel del modelo de datos el recurso DeviceModel:

Nombre	Obligatorio	Tipo	Descripción
id	X	-	Identificador único de la entidad.
type	X	-	Tipo de la entidad. Debe ser igual a DeviceModel
source		Property. Text o URL	Secuencia de caracteres que indican la fuente de los datos de la entidad
dataProvider		Property. URL	Especifica la URL a información sobre el proveedor que informó sobre esta entidad.
category	X	Property. Lista de Text	Las categorías del dispositivo. Valores permitidos, uno de los siguientes o cualquier otro significativo para la aplicación: • **sensor:** un dispositivo que detecta y responde a eventos y cambios en el medio físico, ya sean cambios de luz, movimiento o temperatura. • **actuator**: un dispositivo responsable de mover o controlar un mecanismo o un sistema. • **meter:** un dispositivo construido para detectar de forma precisa una cantidad de algún parámetro y mostrarla en una forma leíble para un humano. • **HVAC:** un dispositivo de calefacción, ventilación y aire acondicionado, que proporciona un ambiente agradable en interior. • **network:** un dispositivo usado para conectar otros dispositivos en una red, ya sea un router un switch o un hub en una red local o red de sensores. • **multimedia:** un dispositivo diseñado para mostrar, almacenar, grabar o reproducir contenido multimedia, ya sea audio, imágenes, animaciones o video. • **implement:** un dispositivo usado o que se necesita para una actividad concreta, herramienta, utensilio, instrumento, etc. • **irrSystem:** un sistema de riego móvil o fijo. • **irrSection:** una sección de un sistema de riego. • **endgun:** un dispositivo acoplado a un sistema de riego encargado de expulsar el agua más allá de él
deviceClass		Property. Text	Clase del dispositivo restringido tal y como se especifica en el RFC 7228. Si el dispositivo no es un dispositivo restringido, esta propiedad no estará presente. Valores permitidos: C0, C1, C2.
controlledProperty	X	Property. Lista de Text	Cualquier cosa que pueda ser medida, sentida o controlada. Valores permitidos: temperature, humidity, light, motion, fillingLevel, occupancy, power, pressure, smoke, energy, airPollution, noiseLevel, weatherConditions, precipitation, windSpeed, windDirection, atmosphericPressure, solarRadiation, depth, pH, conductivity, conductance, tss, tds, turbidity, salinity, orp, cdom, waterPollution, location, speed, heading, weight, waterConsumption, gasComsumption, electricityConsumption, soilMoisture, trafficFlow, eatingActivity, milking, movementActivity o cualquier otro significativo para la aplicación.

function		Property. Lista de Text	La funcionalidad necesaria para llevar a cabo la tarea para la cual un Device está diseñado. Un dispositivo puede ser diseñado para llevar a cabo más de una función. Valores permitidos: levelControl, sensing, onOff, metering, eventNotification.
supportedProtocol		Property. Lista de Text	Protocolos o redes compatibles. Valores permitidos: ul20, mqtt, lwm2m, http, websocket, onem2m, sigfox, lora, nb-iot, ec-gsm-iot, lte-m, cat-m, 3g, grps) o cualquier otro significativo para la aplicación
supportedUnits		Property. Lista de Text	Unidades de medida compatibles con el dispositivo. El código de unidad de medida (texto) indicado por la codificación UN/CEFACT (máximo 3 caracteres).
energyLimitationClass		Property. Text	La clase de limitación energética indicada por el RFC 7228. Valores permitidos: E0, E1, E2, E9
brandName	X	Property. Text	Nombre de la marca del dispositivo
modelName	X	Property. Text	Nombre del modelo del dispositivo
manufacturerName	X	Property. Text	Nombre del fabricante del dispositivo
name	X	Property. Text	Nombre dado a este DeviceModel. Normativa de referencia: https://schema.org/name
description		Property. Text	Descripción del tipo de dispositivo. Normativa de referencia: https://schema.org/description
documentation		Property. URL	Enlace a la documentación del tipo de dispositivo.
image		Property. URL	Enlace a una imagen descriptiva del tipo de dispositivo en cuestión. Normativa de referencia: https://schema.org/image
dateModified		Property. DateTime	Marca de tiempo que indica cuándo se modificó el recurso por última vez. Sólo lectura. Se genera automáticamente.
dateCreated		Property. DateTime	Marca de tiempo que indica cuándo se creó la entidad. Sólo lectura. Se genera automáticamente.

Tabla 2. Recurso DeviceModel de FIWARE


```
{
    "id": "myDevice-wastecontainer-sensor-345",
    "type": "DeviceModel",
    "name": "myDevice Sensor for Containers 345",
    "brandName": "myDevice",
    "modelName": "S4Container 345",
    "manufacturerName": "myDevice Inc.",
    "category": ["sensor"],
    "function": ["sensing"],
    "controlledProperty": ["fillingLevel", "temperature"]
}
```


Figura 39. Recurso DeviceModel de FIWARE en JSON

2.10 FHIR

2.10.1 Recurso Device

Un Device [8], por definición, es un tipo de objeto que se fabrica para prestar asistencia, sin ser cambiado sustancialmente durante su actividad. Puede tener carácter médico o no.

Es un recurso administrativo que representa instancias individuales de un tipo de dispositivo y su localización. Es referenciado por otros recursos, por ejemplo, para dejar constancia de qué dispositivo realizó una acción como puede ser un procedimiento u observación médica. También puede ser referenciado cuando se prescriben dispositivos a un paciente concreto, además es usado para guardar y transmitir el UDI (Unique Device Identifier) asociado a un dispositivo, como puede ser un implante médico. La organización americana FDA [28] (Food and Drugs Administration) es la encargada de establecer este UDI para identificar todos los dispositivos médicos vendidos en los Estados Unidos desde su fabricación. Este identificador no es el que usa este sistema para diferenciar una entidad de otra, sino que utiliza el atributo identifier dentro de Device. Este identificador es más global ya que el sistema UDI solo funciona en los Estados Unidos.

Los recursos que se relacionan directamente con Device son los siguientes,

- Device (este mismo recurso).
- DeviceDefinition - describe un tipo de dispositivo (no es una instancia física).
- DeviceMetric - describe una capacidad de medida, cálculo o configuración de un dispositivo médico.

En FHIR, el Device, al ser un recurso administrativo no cambia mucho y tiene toda la información del fabricante, esto contrasta con el DeviceMetric, el cual modela la parte física, como puede ser el estado operacional del dispositivo, así que es mucho más propenso a cambios.

2.10.2 Recurso DeviceDefinition

Un recurso de tipo DeviceDefinition [9], por definición, describe las características y las capacidades de un dispositivo médico. Se puede decir que es como la definición de catálogo de un dispositivo

La diferencia entre Device y DeviceDefinition es que Device fue pensado para refererirse a instancias físicas de un tipo de dispositivo, por lo tanto, tiene atributos como pueden ser la localización física, número de lote o una referencia a un paciente concreto, los cuáles no son contemplados en DeviceDefinition. En definitiva, Device es un dispositivo concreto y DeviceDefinition proporciona la información del tipo o categoría al que pertenece. Este recurso normalmente es referenciado por un Device.

2.10.3 Recurso DeviceMetric

Como se ha mencionado con anterioridad, un DeviceMetric [10], por definición, describe una capacidad de medida, cálculo o configuración de un dispositivo. Por un lado, describe propiedades estáticas obligatorias que caracterizan, ya sea de forma directa o derivada, cuantitativa o cualitativa, una medida de una señal biológica o un cálculo producido por un dispositivo médico. Por otro, también puede ser usado para describir propiedades no estáticas, pero que son muy importantes para la medida como pueden ser el estado operacional, el modo de medida utilizado, la última fecha en la que fue calibrado, entre otros.

Este tipo de recurso está íntimamente relacionado con Device, así como otro tipo de recurso que recibe el nombre de Observation, que veremos a continuación.

2.10.4 Recurso Observation

Un recurso de tipo Observation [11], por definición, contiene medidas y afirmaciones simples hechas sobre un paciente, dispositivo u otro sujeto.

Las observaciones son un elemento central en el sistema médico, son utilizadas para respaldar un diagnóstico, la monitorización del progreso, para determinar puntos de referencia y patrones, e incluso para capturar características demográficas. La inmensa mayoría de las observaciones son algunos pares de atributos nombre/valor con algo de metadatos, pero también existen observaciones más complejas que agrupan otras observaciones, e incluso observaciones con varios componentes.

Algunos ejemplos de uso del recurso Observation pueden ser, entre otras:

- Señales vitales, como el peso corporal, la presión sanguínea y la temperatura.
- Datos de laboratorio como la glucosa en sangre.
- Medidas fetales.
- Medidas de dispositivos como por ejemplo un pulsioxímetro.
- Características personales, como el color de los ojos.

Un recurso de tipo Observation permite expresar pares nombre-valor o incluso colecciones de pares nombre-valor, aunque pueda almacenar cualquier tipo de información ese no es su objetivo. Fue pensado para capturar medidas y valoraciones médicas en un instante concreto. Tampoco está pensado para contextos más específicos y casos de uso para los que existen ya otros recursos de FHIR más concretos como pueden ser AllergyIntolerance, que representa únicamente las alergias de los pacientes, o MedicationStatement, que modela la medicación que toma un paciente. Estos objetos tan concretos se salen del alcance de este TFG.

2.10.5 HAPI-FHIR

HAPI-FHIR [19] es una implementación de software libre del estándar FHIR en el lenguaje de programación Java. HAPI-FHIR permite el intercambio de información de carácter médico conforme al estándar FHIR.

Es una librería simple, pero muy potente para dotar a la aplicación de la capacidad de manejar y enviar recursos FHIR. Está escrita en Java puro, compatible con la versión 1.6 en adelante.

Podemos encontrar módulos para codificar y decodificar recursos FHIR. Contiene un cliente FHIR que permite recuperar o almacenar recursos en un servidor FHIR externo. Así como módulos para crear servidores conformes al estándar FHIR. Además, modela los recursos del estándar FHIR, proporcionando una interfaz fluida para crear nuevos recursos de forma cómoda y rápida.

2.11 Docker

Docker [29] es una plataforma de software que permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que

incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y entorno de ejecución.

La tecnología Docker usa el kernel de Linux y las funciones de este, como Cgroups y namespaces, para segregar los procesos, de modo que puedan ejecutarse de manera independiente. El propósito de los contenedores es esta independencia: la capacidad de ejecutar varios procesos y aplicaciones por separado para hacer un mejor uso de su infraestructura y, al mismo tiempo, conservar la seguridad que tendría con sistemas separados.

Docker ofrece un modelo de implementación basado en imágenes. Esto permite compartir una aplicación, o un conjunto de servicios, con todas sus dependencias en varios entornos. Docker también automatiza la implementación de las aplicaciones en este entorno de contenedores.

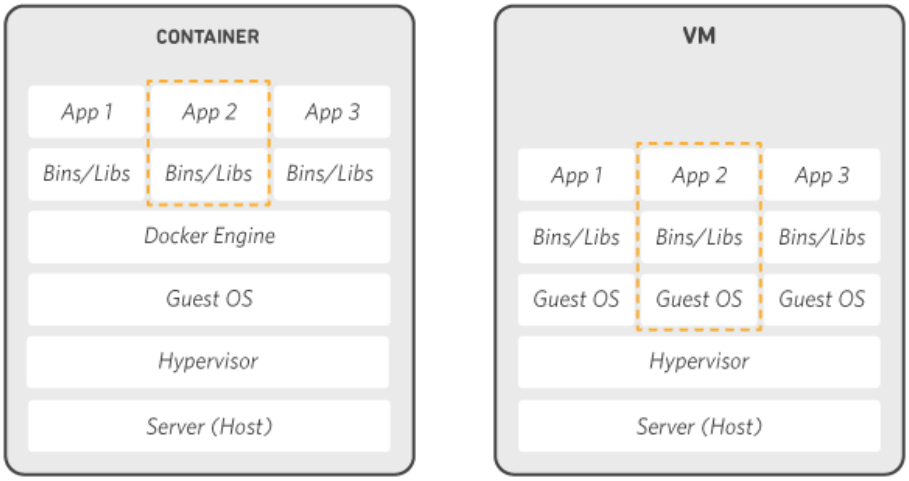

Figura 40. Container de Docker frente a una máquina virtual tradicional

En este proyecto, se va a ejecutar el Orion Context Broker de FIWARE dentro de un contenedor en Docker y en otro se va a ejecutar la base de datos MongoDB que es imprescindible para su funcionamiento.

Se ha decidido hacerlo así porque la instalación de Orion es un proceso bastante complejo ya que tiene un gran número de dependencias. Depende de las librerías libstdc++, boost-thread, boost-filesystem, gnutls, libgcrypt, libcurl, openssl, logrotate y libuuid. Además, recomiendan instalarlo en las distribuciones de linux RedHat o CentOs y este proyecto se ha realizado en Ubuntu. Gracias a Docker, este despliegue tan complejo se resume en un comando.

Además, se explica más adelante la opción de desplegar el servidor FHIR cómodamente usando otro contendor Docker. Así se evita la necesidad de instalar Tomcat en el sistema y la configuración que conlleva.

3 TRABAJO REALIZADO

3.1 Introducción

Como se ha comentado con anterioridad, el objetivo principal de este TFG es el desarrollo de una aplicación que sirva de intermediaria entre dispositivos médicos de la plataforma FIWARE y la plataforma FHIR.

La aplicación ofrece una interfaz REST que permite a un gestor crear, actualizar, recuperar y borrar dispositivos y familias de dispositivos. Una familia de dispositivos tiene como mínimo uno de cada uno de los recursos FHIR con los que trabaja la aplicación. Estos recursos son Device, DeviceDefinition, DeviceMetric y Observation. Al crear los recursos, el gestor tiene que enviarlos en formato JSON y obviamente los recursos tienen que cumplir el estándar FHIR. Para añadir más dispositivos basta con añadir más recursos Device, los otros tres recursos son comunes para toda la familia de dispositivos. Además, gracias a la correspondencia que se especificará en el siguiente apartado, la aplicación crea los recursos Device y DeviceModel de FIWARE con los datos proporcionados según el estándar FHIR. Esto significa que el gestor no tiene que especificar los recursos por separado en FHIR y en FIWARE, sino que sólo los envía una única vez.

Además de la aplicación, se ha lanzado un servidor que maneja recursos FHIR gracias a la libería HAPI FHIR que se mencionó en el capítulo 2. También se ha lanzado una instancia del Orion Context Broker que es el bloque principal de cualquier aplicación FIWARE. Finalmente, se ha creado una base de datos MongoDB que es utilizada tanto por la aplicación como por el Context Broker.

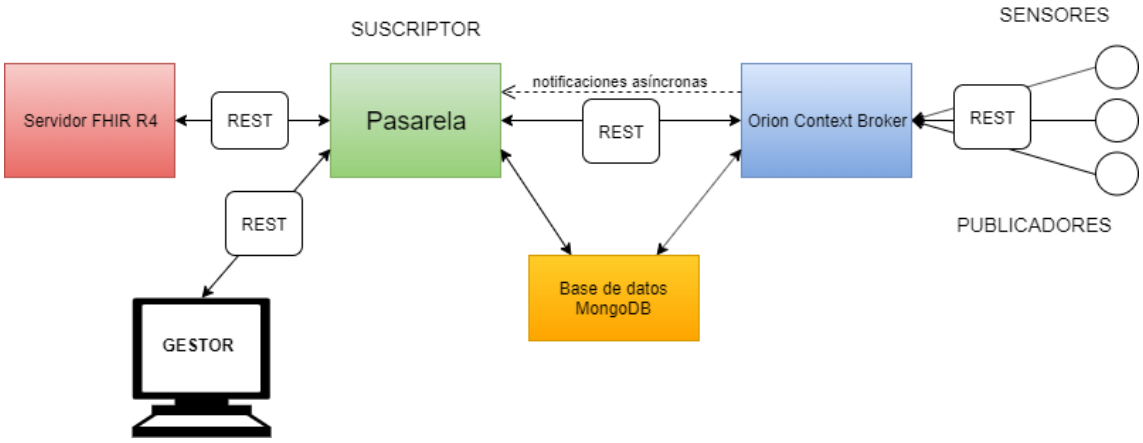

Figura 41. Relaciones de la aplicación con el resto de los componentes

Como se ha explicado en el apartado dedicado a NGSI, en Orion se pueden crear suscripciones muy configurables. Cuando salta una suscripción, se notifica mediante un mensaje HTTP POST a la URLy con los campos que se hayan especificado a la hora de crear la suscripción. Cuando el gestor crea una nueva familia de dispositivos, la aplicación crea las suscripciones necesarias en Orion para que sólo se notifique cuando en el dispositivo cambia el valor de al menos 1 de 5

atributos clave. Estos atributos son el valor medido por el sensor, el estado del sensor, su dirección IP, su posición y la última fecha de calibración. El resto de los atributos tienen un carácter más estático y no van a ser cambiados en la operación normal del sensor.

La aplicación ofrece una segunda interfaz para capturar estas notificaciones HTTP que envía Orion de forma asíncrona. En el cuerpo de estas notificaciones, encontramos nuevos valores para los 5 atributos clave mencionados. La aplicación actualiza los recursos FHIR con estos nuevos valores y envía estos recursos actualizados al servidor FHIR.

La aplicación internamente se divide en 4 componentes diferenciados, un controlador REST y 3 clientes. El controlador REST es el bloque central, para llevar a cabo todas las acciones que ofrece sus dos interfaces usa 3 clientes distintos. El cliente FHIR, el cliente FIWARE y el cliente para trabajar con la base de datos MongoDB. Cada uno de estos bloques se verán en profundidad en este capítulo.

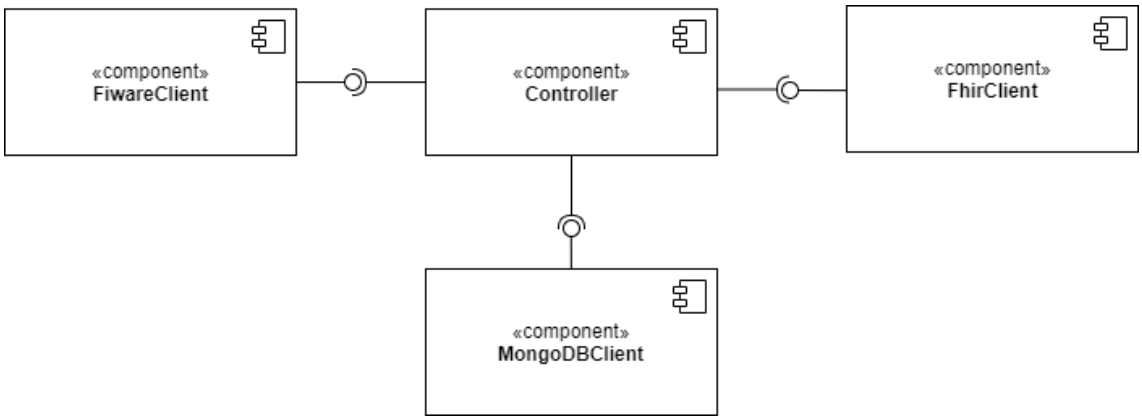

Figura 42. Diagrama de componentes

Al desplegar la aplicación, el servidor FHIR, Orion y la base de datos, el escenario quedaría como muestra el siguiente diagrama. Esto es en el caso de que se despliegue todo en una sola máquina. Se ha hecho así para facilitar el desarrollo de la aplicación. No obstante, al haber usado Docker esto hace que cada uno de los bloques se pueda desplegar en máquinas distintas de forma inmediata. Sólo habría que pasar por la línea de comandos a la hora de ejecutar la aplicación las direcciones IP y puertos correctos. En el siguiente capítulo se detallará como se monta este escenario.

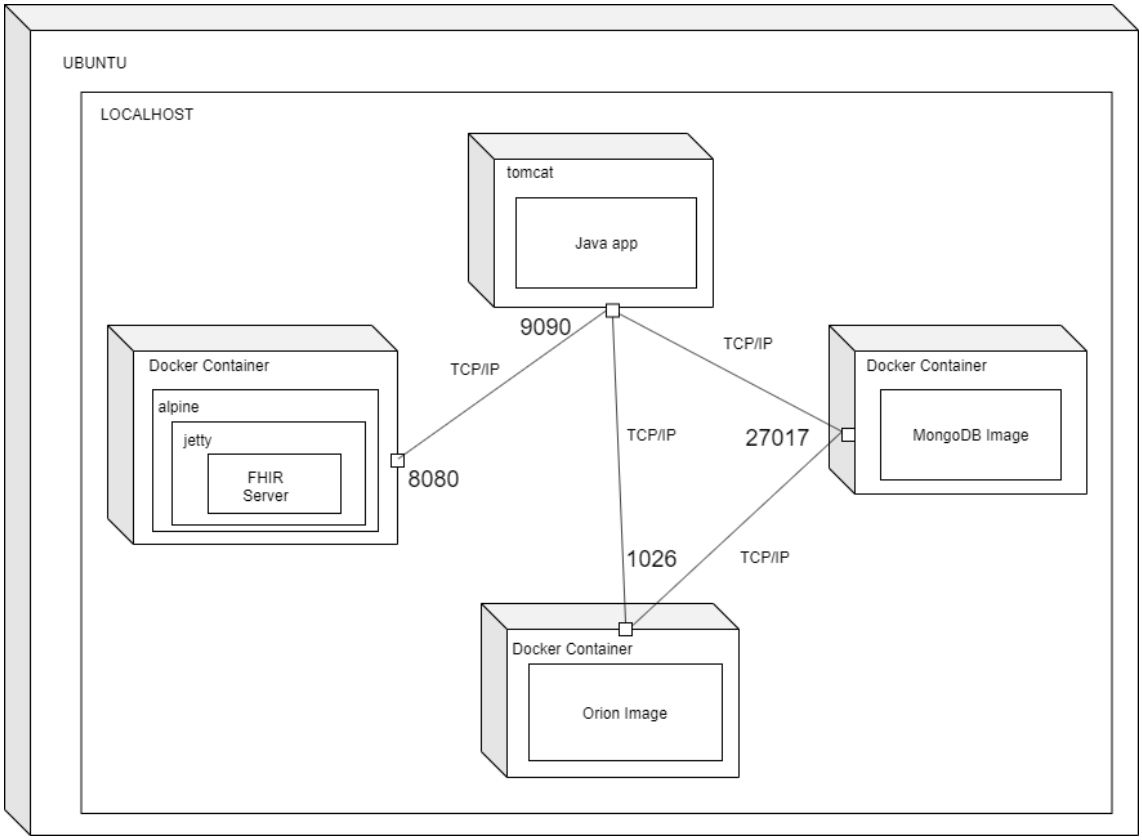

Figura 43. Diagrama de despliegue

3.1.1 Tareas realizadas

- Se ha establecido una correspondencia entre los recursos del estándar FIWARE y FHIR relacionados con los dispositivos en cada plataforma.
- Se ha creado una clase para pasar de FHIR a FIWARE según esta correspondencia.
- Se ha desarrollado una interfaz REST para manejar los dispositivos.
- Se han creado una serie de suscripciones para notificar de los cambios clave.
- Se ha desarrollado una interfaz para capturar las notificaciones asíncronas.
- Se han modelado las suscripciones y entidades FIWARE en java.
- Se han modelado los recursos asociados a dispositivos de FIWARE en java.
- Se ha creado una clase para pasar los recursos FIWARE al estándar NGSIv2.
- Se ha desarrollado un cliente REST para la comunicación con el Context Broker.
- Se ha desarrollado cliente REST para la comunicación con el servidor FHIR R4.
- Se ha desarrollado un cliente para la comunicación con la base de datos MongoDB.
- Se ha desplegado un servidor FHIR R4.
- Se ha desplegado una base de datos MongoDB.
- Se ha desplegado una instancia del Orion Context Broker.
- Se ha desplegado la pasarela sobre un contenedor de servlets.
- Se han realizado una serie de pruebas.

3.2 Correspondencia de recursos FHIR con el ecosistema FIWARE

Como se comentó en la introducción, este estudio se va a centrar en los siguientes recursos de FHIR: Device, DeviceDefinition, DeviceMetric y Observation. A continuación, se van a describir los componentes de cada uno de estos recursos y se va a comentar si existen campos equivalentes en la plataforma FIWARE. Antes de empezar a describir en detalle los componentes de FHIR, recordar que en la plataforma FIWARE se van a utilizar los recursos Device y DeviceModel. La relación entre los recursos quedaría así.

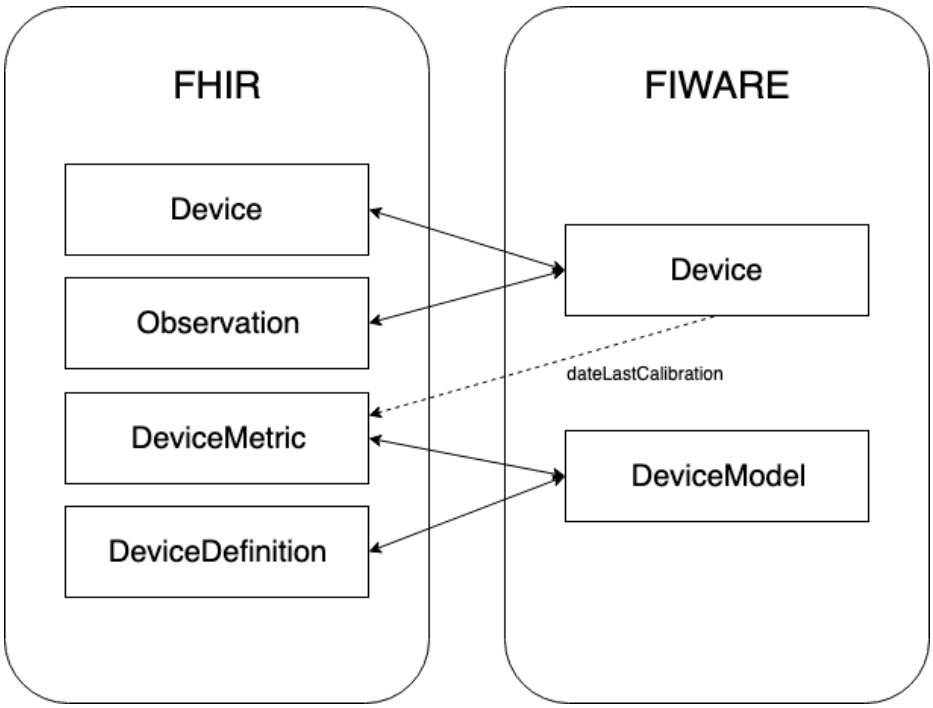

Figura 44. Diagrama de la correspondencia FHIR-FIWARE

3.2.1 DeviceDefinition – Descripción detallada

Hay que recordar que el recurso DeviceDefinition proporciona la información del tipo o categoría al que pertenece un dispositivo médico. Se puede decir que es como la definición de catálogo de un dispositivo.

FHIR: DeviceDefinition.identifier	
Definición	Son uno o varios identificadores únicos de instancia, asignados a un dispositivo ya sea por una herramienta software, el fabricante, otras organizaciones o por los propietarios del dispositivo.
Cardinalidad	0..*
Tipo	Identifier
Notas	Es considerado como un identificador de negocio, no como un identificador lógico del recurso. Esto quiere decir que el identificador lógico del recurso cambia dependiendo del servidor FHIR al que sea enviado, o cambia si un recurso es movido de un lugar a otro. No obstante, el identificador de negocio,

	independientemente del servidor al que sea enviado y de las copias que se hagan, siempre se refiere al mismo recurso, por lo que nos permite mantener la consistencia independientemente del contexto de uso. Todos los recursos FHIR que implementen este identifier, permiten ser buscados por este atributo en cualquier servidor.
FIWARE: DeviceModel	En el estándar FIWARE encontramos un atributo obligatorio de cardinalidad 1, llamado id. Realiza la misma funcionalidad que el identifier de FHIR. No obstante, se diferencian en la cardinalidad, y en la forma de presentar los datos. En FIWARE es un simple string, mientras que en FHIR es un objeto de tipo Identifier con múltiples campos. El id de FIWARE se corresponde con el campo value dentro de Identifier. En FIWARE también encontramos el campo source, este campo opcional indica la fuente de los datos de la entidad, este campo se puede mapear con el campo assigner dentro del tipo Identifier.

Tabla 3. FHIR: DeviceDefinition.identifier

FHIR: DeviceDefinition.udiDeviceIdentifier	
Definición	Es un identificador único de un dispositivo (siglas UDI en inglés, Unique Device identifier) que puede estar asignado a una etiqueta de un dispositivo o paquete. Un dispositivo puede incluir múltiples identificadores UDI, ya que puede incluir uno por cada jurisdicción en la que podría ser vendido el dispositivo.
Cardinalidad	0..*
Tipo	BackboneElement
Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, procederé a examinarlos uno a uno.

Tabla 4. FHIR: DeviceDefinition.udiDeviceIdentifier

FHIR: DeviceDefinition.udiDeviceIdentifier.deviceIdentifier	
Definición	Este es el identificador que se asocia a cada Device que referencie esta DeviceDefinition para el distribuidor y la jurisdicción indicados en el elemento DeviceDefinition.udiDeviceIdentifier .
Cardinalidad	1..1
Tipo	string
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 5. FHIR: DeviceDefinition.udiDeviceIdentifier.deviceIdentifier

FHIR: DeviceDefinition.udiDeviceIdentifier.issuer	
Definición	Indica la organización que asigna el algoritmo de identificación.
Cardinalidad	1..1

Tipo	uri
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 6. FHIR: DeviceDefinition.udiDeviceIdentifier.issuer

FHIR: DeviceDefinition.udiDeviceIdentifier.jurisdiction	
Definición	Indica la jurisdicción a la cual aplica el deviceIdentifier.
Cardinalidad	1..1
Tipo	uri
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 7. FHIR: DeviceDefinition.udiDeviceIdentifier.jurisdiction

FHIR: DeviceDefinition.manufacturer[X]	
Definición	Datos del fabricante
Cardinalidad	0..1
Tipo	Choice [String | Reference (Organization)]
Notas	Hay que elegir entre manufacturerString [String] y manufacturerReference [Reference (Organization)]. Ambas opciones mostradas en las siguientes tablas. No pueden aparecer a la vez.

Tabla 8. FHIR: DeviceDefinition.manufacturer[X]

FHIR: DeviceDefinition.manufacturerString	
Definición	Nombre del fabricante del dispositivo.
Cardinalidad	0..1
Tipo	string
FIWARE DeviceModel	Se corresponde con el campo manufacturerName de DeviceModel. No obstante, el nombre del fabricante también se puede expresar mediante el campo DeviceName.name, si su atributo type toma el valor “manufacturerName”. Se ha decido usar DeviceName por encima de este campo manufacturerString en la correspondencia.

Tabla 9. FHIR: DeviceDefinition.manufacturerString

FHIR: DeviceDefinition.manufacturerReference	
Definición	Referencia a la organización que fabricó el dispositivo.
Cardinalidad	0..1
Tipo	Reference (Organization)

FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 10. FHIR: DeviceDefinition.manufacturerReference

FHIR: DeviceDefinition.deviceName	
Definición	Nombre dado al dispositivo para identificarlo
Cardinalidad	0..*
Tipo	BackboneElement
Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, se procederá a examinarlos uno a uno.

Tabla 11. FHIR: DeviceDefinition.deviceName

FHIR: DeviceDefinition.deviceName.name	
Definición	Nombre del dispositivo
Cardinalidad	1
Tipo	string
Notas	Arriba se ha descrito manufacturerString, que sería equivalente a un nombre indicando el tipo manufacturer-name. Se ha decidido usar mejor este campo.
FIWARE DeviceModel	En FIWARE tenemos varios campos que indican algún tipo de nombre, tenemos modelName, manufacturerName, brandName y name. Serían de los tipos model-name, manufacturer-name, other y user-friendly-name respectivamente. El tipo de nombre se especifica en el siguiente campo.

Tabla 12. FHIR: DeviceDefinition.deviceName.name

FHIR: DeviceDefinition.deviceName.type	
Definición	Tipo del nombre. Puede ser: udi-label-name | user-friendly-name | patient-reported-name | manufacturer-name | model-name | other
Cardinalidad	1..1
Tipo	code
FIWARE DeviceModel	No se corresponde directamente a nada en FIWARE. En FIWARE se puede inferir el tipo del nombre a partir del nombre del atributo como se puede observar en modelName, brandName, etc.

Tabla 13. FHIR: DeviceDefinition.deviceName.type

FHIR: DeviceDefinition.modelNumber	
Definición	El número de modelo del dispositivo
Cardinalidad	0..1

Tipo	string
Notas	Cuidado, no hay que relacionarlo con el atributo de FIWARE modelName, ya que este como se ha indicado arriba se corresponde con un elemento de tipo deviceName.
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 14. FHIR: DeviceDefinition.modelNumber

FHIR: DeviceDefinition.type	
Definición	Indica el tipo de dispositivo con el que estamos trabajando
Cardinalidad	0..1
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE DeviceModel	En DeviceModel existe un atributo con el nombre category que pretende el mismo objetivo final. Lo que pasa es que en FHIR existen más de 1000 tipos diferentes de tipos establecidos como se puede observar en este enlace. https://www.hl7.org/fhir/valueset-device-kind.html. Mientras que en FIWARE sólo existe un número reducido de tipos.

Tabla 15. FHIR: DeviceDefinition.type

FHIR: DeviceDefinition.specialization	
Definición	Define las capacidades y los estándares compatibles con el dispositivo que son usados para la comunicación.
Cardinalidad	0..*
Tipo	BackboneElement
Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, se procederá a examinarlos uno a uno.

Tabla 16. FHIR: DeviceDefinition.specialization

FHIR: DeviceDefinition.specialization.systemType	
Definición	Indica el estándar que se usa para operar y comunicar.
Cardinalidad	1
Tipo	string
FIWARE DeviceModel	En FIWARE existe un atributo con el nombre supportedProtocol, que consiste en una lista de string. Indica todos los estándares usados en la comunicación del dispositivo. Cada elemento de esa lista correspondería con un atributo

	specialization, con el valor de systemType igual al valor del elemento de la lista.

Tabla 17. FHIR: DeviceDefinition.specialization.systemType

FHIR: DeviceDefinition.specialization.version	
Definición	Versión del estándar usado para operar y comunicar.
Cardinalidad	0..1
Tipo	string
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 18. FHIR: DeviceDefinition.specialization.version

FHIR: DeviceDefinition.version	
Definición	Indica el historial de versiones disponibles del dispositivo, por ejemplo, las versiones software que existen.
Cardinalidad	0..*
Tipo	string
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 19. FHIR: DeviceDefinition.version

FHIR: DeviceDefinition.safety	
Definición	Características de seguridad del dispositivo.
Cardinalidad	0..*
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 20. FHIR: DeviceDefinition.safety

FHIR: DeviceDefinition.shelfLifeStorage	
Definición	Información de almacenaje y caducidad del dispositivo. Se usa con productos médicos o con contenedores de estos productos.
Cardinalidad	0..*
Tipo	ProductShelfLife

Notas	El tipo ProductShelfLife es a su vez un BackBoneElement, como se ha comentado, eso significa que tiene una serie de hijos. Este objeto mediante sus parámetros describe el tiempo de vida que tiene el producto médico, las precauciones especiales que hay que tener en cuenta para su almacenaje e instrucciones relacionadas con el mantenimiento del producto, entre otros.
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 21. FHIR: DeviceDefinition.shelfLifeStorage

FHIR: DeviceDefinition.physicalCharacteristics	
Definición	Define propiedades físicas del dispositivo como sus dimensiones, su color, su volumen y su forma.
Cardinalidad	0..1
Tipo	ProdCharacteristic
Notas	El tipo ProdCharacteristic es a su vez un BackboneElement. Tiene atributos que permiten indicar su altura, anchura, profundidad, peso, volumen, diámetro, forma y color. Tiene incluso un atributo para adjuntar una imagen del producto.
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 22. FHIR: DeviceDefinition.physicalCharacteristics

FHIR: DeviceDefinition.languageCode	
Definición	Idioma en el que se muestran los textos producidos por el dispositivo.
Cardinalidad	0..*
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 23. FHIR: DeviceDefinition.languageCode

FHIR: DeviceDefinition.capability	
Definición	Capacidades del dispositivo
Cardinalidad	0..*
Tipo	BackboneElement
Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, se procederá a examinarlos uno a uno.

Tabla 24. FHIR: DeviceDefinition.capability

FHIR: DeviceDefinition.capability.type	
Definición	Tipo de la capacidad
Cardinalidad	1
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE DeviceModel	En FIWARE se observa un campo con el nombre de function, que es una lista de string que indica las funcionalidades que implementa el dispositivo, necesarias para cumplir su fin. Por ejemplo, una funcionalidad puede ser medir un valor o notificar algún suceso. Dado que DeviceDefinition es un recurso que sólo aparece en la versión R4 de FHIR, no se han encontrado ejemplos de un uso específico del campo capability de FHIR. Por lo tanto, con los datos en mano, se ha decidido que el valor de function en FIWARE se podría relacionar con este elemento type.

Tabla 25. FHIR: DeviceDefinition.capability.type

FHIR: DeviceDefinition.capability.description	
Definición	Descripción de la capacidad
Cardinalidad	0..*
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 26. FHIR: DeviceDefinition.capability.description

FHIR: DeviceDefinition.property	
Definición	Indica las opciones de configuración de un dispositivo en su operación.
Cardinalidad	0..*
Tipo	BackboneElement
Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, se procederá a examinarlos uno a uno.

Tabla 27. FHIR: DeviceDefinition.property

FHIR: DeviceDefinition.property.type	
Definición	Código que especifica de qué propiedad se está hablando
Cardinalidad	1..1
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico. Aquí encontramos una serie de códigos asociados a propiedades concretas. https://www.hl7.org/fhir/valueset-device-component-property.html
FIWARE: DeviceModel	En FIWARE estas opciones de configuración del dispositivo se modelan dentro del Device de FIWARE. Dado que este mismo atributo property se repite en el recurso Device de FHIR. Se ha considerado sólo incluir la información dentro de este último. Ya que tiene más sentido que esta configuración esté en cada dispositivo, y no en un tipo más genérico como este.

Tabla 28. FHIR: DeviceDefinition.property.type

FHIR: DeviceDefinition.property.valueQuantity	
Definición	El valor de la propiedad como cantidad
Cardinalidad	0..*
Tipo	Quantity
Notas	El tipo Quantity sirve para describir una cantidad que haya sido medida o que se pueda medir. Tiene campos que describen el valor, su unidad, su forma codificada y el sistema en el que se codifica.
FIWARE: DeviceModel	En FIWARE estas opciones de configuración del dispositivo se modelan dentro del Device de FIWARE. Dado que este mismo atributo property se repite en el recurso Device de FHIR. Se ha considerado sólo incluir la información dentro de este último. Ya que tiene más sentido que esta configuración esté en cada dispositivo, y no en un tipo más genérico como este.

Tabla 29. FHIR: DeviceDefinition.property.valueQuantity

FHIR: DeviceDefinition.property.valueCode	
Definición	El valor de la propiedad como código
Cardinalidad	0..*
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 30. FHIR: DeviceDefinition.property.valueCode

FHIR: DeviceDefinition.owner	
Definición	Referencia a la organización responsable del dispositivo
Cardinalidad	0..1
Tipo	Reference (Organization)
Notas	El recurso Reference, tiene campos que permiten indicar cualquier tipo de referencia, ya sea literal, relativa, interna o una URL absoluta. Así como una referencia lógica cuando no se sabe la referencia literal. Incluso contiene un campo para indicar un texto alternativo en caso de que no haya ninguna referencia a otro recurso.
FIWARE: DeviceModel	Este campo aparece en el recurso Device de FIWARE con el mismo nombre, owner. En este caso implementa una lista de referencias, no solo a una o varias Organizaciones, sino también a Personas. Otra opción de implementación alternativa en FIWARE sería una lista de URIs. Por lo tanto, aquí solo se podría hacer una correspondencia en caso de la referencia a una Organización. No obstante, este campo owner aparece también en el recurso Device de FHIR. Se ha decidido que tiene más sentido mapearlo con ese campo para mantener la relación entre los recursos Device de ambas plataformas.

Tabla 31. FHIR: DeviceDefinition.owner

FHIR: DeviceDefinition.contact	
Definición	Información de contacto de una organización o persona particular responsable del dispositivo. Normalmente se utiliza como respaldo si el dispositivo tuviera un problema.
Cardinalidad	0..*
Tipo	ContactPoint
Notas	El tipo ContactPoint contiene varios campos, entre los que encontramos uno llamado system para indicar el sistema de comunicación (fax, teléfono, página web, correo electrónico, ...), otro llamado value para indicar el punto de contacto en sí. También permite establecer un periodo de validez, un rango de preferencia y una etiqueta asociada al uso del punto de comunicación.
FIWARE DeviceModel	En FIWARE podemos encontrar un campo que recibe el nombre de dataProvider, es un atributo opcional que se modela como una URL. Especifica una dirección que apunta al responsable de la información acerca del dispositivo. Sería una versión muy simplificada del ContactPoint.

Tabla 32. FHIR: DeviceDefinition.contact

FHIR: DeviceDefinition.url	
Definición	Dirección de red del dispositivo
Cardinalidad	0..1

Tipo	uri
Notas	Si el dispositivo está ejecutando un servidor FHIR, la dirección de red debería ser la URL base desde la cual la declaración de conformidad pueda ser recuperada.
FIWARE: DeviceModel	El recurso Device de FIWARE cuenta con un campo similar que recibe el nombre de ipAddress. No obstante, mientras que aquí la dirección solo puede ser una, en FIWARE es una lista de direcciones IP. Dado que este campo también aparece en el Device de FHIR, se va a relacionar con ese en lugar de este campo url de DeviceDefinition, ya que las direcciones IP se asignan a los dispositivos concretos.

Tabla 33. FHIR: DeviceDefinition.url

FHIR: DeviceDefinition.onlineInformation	
Definición	Dirección que permite acceder a información en línea sobre el dispositivo
Cardinalidad	0..1
Tipo	uri
FIWARE DeviceModel	En FIWARE encontramos el campo documentation, que es opcional, y se modela como una URL. Es un link a la documentación del dispositivo. Sería una correspondencia directa.

Tabla 34. FHIR: DeviceDefinition.onlineInformation

FHIR: DeviceDefinition.note	
Definición	Una serie de notas y comentarios descriptivos acerca del dispositivo
Cardinalidad	0..*
Tipo	Annotation
Notas	El tipo Annotation consta de tres campos, uno que indica el autor de la nota, otro su contenido en forma de texto y finalmente uno que nos informa de cuándo se hizo la nota.
FIWARE DeviceModel	En FIWARE, encontramos el campo description, es simplemente un campo de texto para describir un dispositivo. Sería una versión muy simplificada de campo note, ya que note aparte de indicar la fecha y el autor, permite almacenar varias notas, a diferencia del campo de FIWARE que sólo permite una única descripción.

Tabla 35. FHIR: DeviceDefinition.note

FHIR: DeviceDefinition.quantity	
Definición	La cantidad de dispositivos presente en el paquete, por ejemplo, el número de dispositivos presentes dentro de un pack, o el número de dispositivos en el mismo paquete de un producto médico
Cardinalidad	0..1

Tipo	Quantity
Notas	El tipo Quantity sirve para describir una cantidad que haya sido medida o que se pueda medir. Tiene campos que describen el valor, su unidad, su forma codificada y el sistema en el que se codifica.
FIWARE: DeviceModel	No se corresponde a nada en FIWARE.

Tabla 36. FHIR: DeviceDefinition.quantity

FHIR: DeviceDefinition.parentDevice	
Definición	El dispositivo padre del cual puede formar parte
Cardinalidad	0..1
Tipo	Reference (DeviceDefinition)
Notas	El recurso Reference, tiene campos que permiten indicar cualquier tipo de referencia, ya sea literal, relativa, interna o una URL absoluta. También puede indicar el tipo del recurso. Así como una referencia lógica cuando no se sabe la referencia literal. Incluso contiene un campo para indicar un texto alternativo en caso de que no haya ninguna referencia a otro recurso.
FIWARE: DeviceModel	En FIWARE no existe esta capacidad de crear modelos tan complejos con dispositivos padres de otros dispositivos.

Tabla 37. FHIR: DeviceDefinition.parentDevice

FHIR: DeviceDefinition.material	
Definición	Indica una sustancia usada en la creación de los materiales de los que está hecho el dispositivo.
Cardinalidad	0..*
Tipo	BackboneElement
Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, se procederá a examinarlos uno a uno.

Tabla 38. FHIR: DeviceDefinition.material

FHIR: DeviceDefinition.material.substance	
Definición	La sustancia en sí
Cardinalidad	1
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.

FIWARE: DeviceModel	No se corresponde a nada en FIWARE

Tabla 39. FHIR: DeviceDefinition.material.substance

FHIR: DeviceDefinition.material.alternate	
Definición	Indica si hay un material alternativo para el dispositivo
Cardinalidad	0..1
Tipo	boolean
FIWARE	No se corresponde a nada en FIWARE

Tabla 40. FHIR: DeviceDefinition.material.alternate

FHIR: DeviceDefinition.material.allergenicIndicator	
Definición	Indica si la sustancia es un alérgeno conocido o sospechado
Cardinalidad	0..1
Tipo	boolean
FIWARE: DeviceModel	No se corresponde a nada en FIWARE

Tabla 41. FHIR: DeviceDefinition.material.allergenicIndicator

3.2.2 Device – Descripción detallada

Como se ha mencionado con anterioridad, un Device es un recurso administrativo que representa instancias individuales de un tipo de dispositivo y su localización. Es referenciado por otros recursos, por ejemplo, para dejar constancia de qué dispositivo realizó una acción como puede ser un procedimiento u observación médica.

FHIR: Device.identifier	
Definición	Son uno o varios identificadores únicos de instancia, asignados a un dispositivo ya sea por una herramienta software, el fabricante, otras organizaciones o por los propietarios del dispositivo.
Cardinalidad	0..*
Tipo	Identifier
Notas	Es considerado como un identificador de negocio, no como un identificador lógico del recurso. Esto quiere decir que el identificador lógico del recurso cambia dependiendo del servidor FHIR al que sea enviado, o cambia si un recurso es movido de un lugar a otro. No obstante el identificador de negocio, independientemente del servidor al que sea enviado y de las copias que se hagan, siempre se refiere al mismo recurso, por lo que nos permite mantener la consistencia independientemente del contexto de uso. Todos los recursos FHIR

	que implementen este identifier, permiten ser buscados por este atributo en cualquier servidor.
FIWARE Device	En el estándar FIWARE encontramos un atributo obligatorio de cardinalidad 1, llamado id. Realiza la misma funcionalidad que el identifier de FHIR. Se diferencian en la cardinalidad, y en la forma de presentar los datos. En FIWARE es un simple string, mientras que en FHIR es un objeto de tipo Identifier con múltiples campos. El id de FIWARE se corresponde con el campo value dentro de Identifier.

Tabla 42. FHIR: Device.identifier

FHIR: Device.definition	
Definición	Es la referencia a la definición (DeviceDefinition) correspondiente de este dispositivo.
Cardinalidad	0..1
Tipo	Reference (DeviceDefinition)
Notas	El recurso Reference, tiene campos que permiten indicar cualquier tipo de referencia, ya sea literal, relativa, interna o una URL absoluta. También puede indicar el tipo del recurso. Así como una referencia lógica cuando no se sabe la referencia literal. Incluso contiene un campo para indicar un texto alternativo en caso de que no haya ninguna referencia a otro recurso.
FIWARE Device	En FIWARE encontramos el campo refDeviceModel, es un campo opcional, modelado como una referencia de FIWARE. Realiza la misma función exactamente, así que sería una correspondencia directa.

Tabla 43. FHIR: Device.definition

FHIR: Device.udiCarrier	
Definición	Es un identificador único de un dispositivo (siglas UDI en Inglés, Unique Device identifier) que puede estar asignado a una etiqueta de un dispositivo o paquete. Un dispositivo puede incluir múltiples identificadores UDI, ya que puede incluir una por cada jurisdicción en la que podría ser vendido el dispositivo.
Cardinalidad	0..*
Tipo	BackboneElement
Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, se procederá a examinarlos uno a uno. El UDI puede identificar una única instancia de un dispositivo, o también puede que solo identifique el tipo del dispositivo.
FIWARE Device	Ningún campo se corresponde con nada en FIWARE.

Tabla 44. FHIR: Device.udiCarrier

FHIR: Device.udiCarrier.deviceIdentifier	
Definición	Es una porción fija y obligatoria de un UDI, que identifica quién ha etiquetado el producto y el modelo o versión específica del dispositivo. El nombre abreviado de este campo es DI.
Cardinalidad	0..1
Tipo	string

Tabla 45. FHIR: Device.udiCarrier.deviceIdentifier

FHIR: Device.udiCarrier.issuer	
Definición	Es la organización encargada de administrar los UDIs para los dispositivos.
Cardinalidad	0..1
Tipo	uri
Notas	Algunos ejemplos de organizaciones son: • GS1: http://hl7.org/fhir/NamingSystem/gs1-di • HIBCC: http://hl7.org/fhir/NamingSystem/hibcc-dI • ICCBBA para contenedores de sangre: http://hl7.org/fhir/NamingSystem/iccbba-blood-di • ICCBA para otros dispositivos: http://hl7.org/fhir/NamingSystem/iccbba-other-di

Tabla 46. FHIR: Device.udiCarrier.issuer

FHIR: Device.udiCarrier.jurisdiction	
Definición	Es la identidad de la fuente autorizada para generar UDIs dentro de una jurisdicción. Todas las UDIs son únicas globalmente dentro de un único espacio de nombres.
Cardinalidad	0..1
Tipo	uri
Notas	Permite saber qué base de datos contendrá todos los metadatos asociados. Por ejemplo, para las UDIs de los dispositivos manejados dentro del ámbito americano, por la FDA (Food and Drug Administration), el valor sería: http://hl7.org/fhir/NamingSystem/fda-udi.

Tabla 47. FHIR: Device.udiCarrier.jurisdiction

FHIR: Device.udiCarrier.carrierAIDC	
Definición	Es la representación del código de barras UDI impreso en el empaquetado del dispositivo en una forma que solo entienden las máquinas. Lo leen usando la tecnología AIDC [30](Automatic Identification and Data Capture). En el siguiente campo se encuentra la versión que puede ser leída por los humanos.

Cardinalidad	0..1
Tipo	base64Binary
Notas	Debido a las limitaciones de los juegos de caracteres en XML, los formatos AIDC DEBERÁN estar codificados en base64. Actualmente no existe un límite máximo para el tamaño de la codificación, pero los sistemas tendrán que imponer un valor máximo en la implementación final, debiendo quedar esto documentado.

Tabla 48. FHIR: Device.udiCarrier.carrierAIDC

FHIR: Device.udiCarrier.carrierHRF	
Definición	Es la representación del código de barras UDI impreso en el empaquetado del dispositivo en la forma HRF (Human Readable Form).
Cardinalidad	0..1
Tipo	string

Tabla 49. FHIR: Device.udiCarrier.carrierHRF

FHIR: Device.udiCarrier.entryType	
Definición	Indica de qué forma fueron registrados los datos en el sistema. Permite distinguir si fueron introducidos por un humano o una máquina. Los códigos permitidos son barcode, rfid y manual entre otros. Aquí está el listado completo: https://www.hl7.org/fhir/valueset-udi-entry-type.html
Cardinalidad	0..1
Tipo	code

Tabla 50. FHIR: Device.udiCarrier.entryType

FHIR: Device.status	
Definición	Estado de disponibilidad del recurso Device. Los códigos posibles para este campo son active, inactive, entered-in-error y unkown.
Cardinalidad	0..1
Tipo	code
FIWARE Device	En FIWARE encontramos un campo opcional con el nombre deviceState que tiene exactamente la misma utilidad. La diferencia radica en que en FIWARE se modela el campo con un simple texto, que puede tomar cualquier valor. En FHIR, al estar representado por un code, por lo tanto solo puede tomar una serie de valores predefinidos.

Tabla 51. FHIR: Device.status

FHIR: Device.statusReason	
Definición	Define la razón del estado de disponibilidad del campo Device.status .

	Los posibles valores son: online, paused, standby, offline, not-ready, trasnduc-discon, hw-discon y off.
Cardinalidad	0..*
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE Device	No se corresponde a nada en FIWARE

Tabla 52. FHIR: Device.statusReason

FHIR: Device.distinctIdentifier	
Definición	Es un identificador distintivo requerido por algunas regulaciones, como puede ser la regulación federal americana (CFR [31]). Aplica tanto a células, o tejidos humanos, como a productos basados en ellos.
Cardinalidad	0..1
Tipo	string
Notas	Su abreviatura es DIC de Distinct Identification Code
FIWARE Device	No se corresponde a nada en FIWARE

Tabla 53. FHIR: Device.distinctIdentifier

FHIR: Device.manufacturer	
Definición	Nombre del fabricante del dispositivo
Cardinalidad	0..1
Tipo	string
FIWARE Device	En Device de FIWARE existe un campo opcional llamado provider que indica el fabricante del dispositivo. Así que sería una correspondencia directa.

Tabla 54. FHIR: Device.manufacturer

FHIR: Device.manufacturerDate	
Definición	La fecha y la hora en la que el dispositivo fue fabricado
Cardinalidad	0..1
Tipo	dateTime
Notas	En FHIR el formato es el siguiente: YYYY, YYYY-MM, YYYY-MM-DD o YYYY-MM-DDThh:mm:ss+zz:zz

FIWARE Device	Encontramos una correspondencia directa con el campo opcional dateManufactured de FIWARE. Tiene la misma función. En algún caso habrá que adaptar el formato de la fecha. En FIWARE es: [-]YYYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm], según la ISO 8601.

Tabla 55. FHIR: Device.manufacturerDate

FHIR: Device.expirationDate	
Definición	La fecha y la hora a partir de la cual el dispositivo no es válido, o no debería ser usado.
Cardinalidad	0..1
Tipo	dateTime
FIWARE Device	No se corresponde a nada en FIWARE

Tabla 56. FHIR: Device.expirationDate

FHIR: Device.lotNumber	
Definición	Número de lote asignado por el fabricante
Cardinalidad	0..1
Tipo	string
FIWARE Device	No se corresponde a nada en FIWARE

Tabla 57. FHIR: Device.lotNumber

FHIR: Device.serialNumber	
Definición	Número de serie asignado por el fabricante
Cardinalidad	0..1
Tipo	string
FIWARE Device	Hay una correspondencia directa con el campo opcional serialNumber de FIWARE.

Tabla 58. FHIR: Device.serialNumber

FHIR: Device.deviceName	
Definición	Este campo representa el nombre del dispositivo, puede ser proporcionado por el mismo dispositivo, una etiqueta UDI o una persona que describa el dispositivo. Normalmente este campo se usa cuando una persona le da un nombre o nombres, o cuando el dispositivo representa uno de los campos mencionados en su DeviceDefinition.
Cardinalidad	0..*
Tipo	BackboneElement

Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, se procederá a examinarlos uno a uno.

Tabla 59. FHIR: Device.deviceName

FHIR: Device.deviceName.name	
Definición	El nombre del dispositivo
Cardinalidad	1
Tipo	string
Notas	En DeviceModel hay más tipos de nombres, pero estos ya se han relacionado con DeviceDefinition.
FIWARE Device	Hay una correspondencia directa con el campo name de FIWARE. Tener en cuenta que mientras que en FIWARE solo se permite un nombre, aquí se pueden asociar varios nombres, mediante un deviceName cada uno.

Tabla 60. FHIR: Device.deviceName.name

FHIR: Device.deviceName.type	
Definición	Tipo del nombre. Puede ser UDILabelName | UserFriendlyName | PatientReportedName | ManufactureDeviceName | ModelName.
Cardinalidad	1
Tipo	code
FIWARE Device	No se corresponde a ningún campo en FIWARE. Por lo que se puede leer en la documentación de FIWARE, el tipo del nombre mencionado arriba sería UserFriendlyName.

Tabla 61. FHIR: Device.deviceName.type

FHIR: Device.modelNumber	
Definición	Número de modelo del dispositivo
Cardinalidad	0..1
Tipo	string
FIWARE Device	No se corresponde a ningún campo en FIWARE

Tabla 62. FHIR: Device.modelNumber

FHIR: Device.partNumber	
Definición	Número de identificador de una parte concreta dentro del dispositivo
Cardinalidad	0..1
Tipo	string

FIWARE Device	No se corresponde a ningún campo en FIWARE

Tabla 63. FHIR: Device.partNumber

FHIR: Device.type	
Definición	La familia o tipo de un dispositivo
Cardinalidad	0..1
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE Device	En Device de FIWARE existe un atributo opcional, que también aparece en DeviceModel, con el nombre category que pretende el mismo objetivo final. Lo que pasa que en FHIR existen más de 1000 tipos diferentes de tipos establecidos como se puede observar en este enlace. https://www.hl7.org/fhir/valueset-device-kind.html. Mientras que en FIWARE sólo existe un número reducido de tipos.

Tabla 64. FHIR: Device.type

FHIR: Device.specialization	
Definición	Define las capacidades y los estándares compatibles con el dispositivo que son usados para la comunicación.
Cardinalidad	0..*
Tipo	BackboneElement
Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, se procederá a examinarlos uno a uno.

Tabla 65. FHIR: Device.specialization

FHIR: Device.specialization.systemType	
Definición	Indica el estándar que se usa para operar y comunicar.
Cardinalidad	1
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE Device	En DeviceModel de FIWARE existe un atributo con el nombre supportedProtocol que consiste en una lista de string. Este indica todos los estándares usados en la comunicación del dispositivo. Se estableció una correspondencia con el campo specialization de DeviceDefinition. En el recurso

	Device de FIWARE vuelve a aparecer el campo supportedProtocol pero solo se recomienda usarlo si, debido a una actualización de software, se ha añadido un nuevo protocolo de comunicación. En otro caso debería bastar con el campo de DeviceModel. Por lo tanto, solo se establecería una correspondencia de este campo con supportedProtocol si el protocolo mencionado no aparece en DeviceModel ni DeviceDefinition. Cada elemento de esa lista correspondería con un atributo specialization, con el valor de systemType igual al valor del elemento de la lista.

Tabla 66. FHIR: Device.specialization.systemType

FHIR: Device.specialization.version	
Definición	Versión del estándar que es usado para operar y comunicar.
Cardinalidad	0..1
Tipo	string
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 67. FHIR: Device.specialization.version

FHIR: Device.version	
Definición	Indica la versión, ya sea hardware o software, que se está usando actualmente en el dispositivo.
Cardinalidad	0..*
Tipo	BackboneElement
Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, se procederá a examinarlos uno a uno.

Tabla 68. FHIR: Device.version

FHIR: Device.version.type	
Definición	Tipo de la versión del dispositivo.
Cardinalidad	0..1
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE Device	No se corresponde a ningún campo en FIWARE, el tipo de la versión se deduce del nombre del campo. En Device tenemos 4 campos que se refieren a versiones: hardwareVersion, softwareVersion, firmwareVersion y osVersion.

Tabla 69. FHIR: Device.version.type

FHIR: Device.version.component	
Definición	Componente único de la versión del dispositivo.
Cardinalidad	0..1
Tipo	Identifier
FIWARE Device	No se corresponde a ningún campo en FIWARE

Tabla 70. FHIR: Device.version.component

FHIR: Device.version.value	
Definición	Texto en el que se indica la versión
Cardinalidad	1..1
Tipo	string
FIWARE Device	En Device de FIWARE tenemos 4 campos que se refieren a versiones: hardwareVersion, softwareVersion, firmwareVersion y osVersion. Para saber con qué campo establecer la correspondencia, habrá que comprobar el campo Device.version.type de FHIR. Por ejemplo, si el valor de Device.version.type fuera "Hardware" el valor de Device.version.value se asignaría al campo de FIWARE Device.hardwareVersion.

Tabla 71. FHIR: Device.version.value

FHIR: Device.property	
Definición	Indica las opciones de configuración de un dispositivo en su operación
Cardinalidad	0..*
Tipo	BackboneElement
Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, se procederá a examinarlos uno a uno.

Tabla 72. FHIR: Device.property

FHIR: Device.property.type	
Definición	Código que especifica de qué propiedad estamos hablando
Cardinalidad	1..1
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.

FIWARE Device	En Device tenemos el campo configuration, que es un mapa que asocia propiedades con sus parámetros. La llave del mapa estaría asociada con este campo type y el valor asociado se mapearía con valueCode.

Tabla 73. FHIR: Device.property.type

FHIR: Device.property.valueQuantity	
Definición	El valor de la propiedad como cantidad
Cardinalidad	0..*
Tipo	Quantity
FIWARE Device	No se corresponde a ningún campo en FIWARE

Tabla 74. FHIR: Device.property.valueQuantity

FHIR: Device.property.valueCode	
Definición	El valor de la propiedad como código
Cardinalidad	0..*
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE Device	En Device tenemos el campo configuration, que es un mapa que asocia propiedades con sus parámetros. La llave del mapa sería el tipo y el valor asociado lo mapearíamos con valueCode.

Tabla 75. FHIR: Device.property.valueCode

FHIR: Device.patient	
Definición	Información del paciente, en el caso de que el dispositivo esté asociado a una persona.
Cardinalidad	0..1
Tipo	Reference (Patient)
Notas	El recurso Reference, tiene campos que permiten indicar cualquier tipo de referencia, ya sea literal, relativa, interna o una URL absoluta. Así como una referencia lógica cuando no se sabe la referencia literal. Incluso contiene un campo para indicar un texto alternativo en caso de que no haya ninguna referencia a otro recurso.
FIWARE Device	En el recurso Device de FIWARE existe un campo con el nombre de owner. Su implementación consiste en una lista de referencias tanto a organizaciones como a personas. Por lo tanto, en el caso de ser una referencia a una persona se podría mapear con este campo Device.Patient.

Tabla 76. FHIR: Device.patient

FHIR: Device.owner	
Definición	Referencia a la organización responsable del suministro y mantenimiento del dispositivo.
Cardinalidad	0..1
Tipo	Reference (Organization)
Notas	El recurso Reference, tiene campos que permiten indicar cualquier tipo de referencia, ya sea literal, relativa, interna o una URL absoluta. Así como una referencia lógica cuando no se sabe la referencia literal. Incluso contiene un campo para indicar un texto alternativo en caso de que no haya ninguna referencia a otro recurso.
FIWARE Device	En el recurso Device de FIWARE existe un campo con el nombre de owner. Su implementación consiste en una lista de referencias tanto a organizaciones como a personas. Por lo tanto, en el caso de ser una referencia a una organización se podría mapear con este campo Device.owner.

Tabla 77. FHIR: Device.owner

FHIR: Device.contact	
Definición	Información de contacto de una organización o persona particular responsable del dispositivo. Normalmente se utiliza como respaldo si el dispositivo tuviera un problema.
Cardinalidad	0..*
Tipo	ContactPoint
Notas	El tipo ContactPoint contiene varios campos, entre los que encontramos uno llamado system para indicar el sistema de comunicación (fax, teléfono, página web, correo electrónico, …), otro llamado value para indicar el punto de contacto en sí. También permite establecer un periodo de validez, un rango de preferencia y una etiqueta asociada al uso del punto de comunicación.
FIWARE Device	En FIWARE podemos encontrar un campo que recibe el nombre de dataProvider, es un atributo opcional que se modela como una URL. Especifica una dirección que apunta al responsable de la información acerca del dispositivo. Sería una versión muy simplificada del ContactPoint.

Tabla 78. FHIR: Device.contact

FHIR: Device.location	
Definición	Indica el lugar donde puede ser encontrado el dispositivo
Cardinalidad	0..1
Tipo	Reference (Location)
Notas	El recurso Reference, tiene campos que permiten indicar cualquier tipo de referencia, ya sea literal, relativa, interna o una URL absoluta. También puede indicar el tipo del recurso. Así como una referencia lógica cuando no se sabe la

	referencia literal. Incluso contiene un campo para indicar un texto alternativo en caso de que no haya ninguna referencia a otro recurso.
FIWARE Device	En FIWARE encontramos el campo con el nombre también de location. La diferencia radica que en FIWARE se modela la posición mediante el tipo GeoJson. Por lo tanto, es una correspondencia directa, salvando la conversión del tipo Location a GeoJson.

Tabla 79. FHIR: Device.location

FHIR: Device.url	
Definición	Dirección de red del dispositivo
Cardinalidad	0..1
Tipo	uri
Notas	Si el dispositivo está ejecutando un servidor FHIR, la dirección de red debería ser la URL base desde la cual la declaración de conformidad pueda ser recuperada.
FIWARE Device	El recurso Device de FIWARE cuenta con un campo similar que recibe el nombre de ipAddress. No obstante, mientras que aquí a la dirección solo puede ser una, en FIWARE es una lista de direcciones IP.

Tabla 80. FHIR: Device.url

FHIR: Device.note	
Definición	Una serie de notas y comentarios que contienen información descriptiva acerca del dispositivo, así como información de uso o implementación.
Cardinalidad	0..*
Tipo	Annotation
Notas	El tipo Annotation consta de tres campos, uno que indica el autor de la nota, otro su contenido en forma de texto y finalmente uno que nos informa de cuándo se hizo la nota.
FIWARE Device	En FIWARE, encontramos el campo description, es simplemente un campo de texto para describir un dispositivo. Sería una versión muy simplificada de campo note, ya que note aparte de indicar la fecha y el autor, permite almacenar varias notas, a diferencia del campo de FIWARE que sólo permite una única descripción.

Tabla 81. FHIR: Device.note

FHIR: Device.safety	
Definición	Características de seguridad del dispositivo.
Cardinalidad	0..*
Tipo	CodeableConcept

Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE	No se corresponde a nada en FIWARE

Tabla 82. FHIR: Device.safety

FHIR: Device.parent	
Definición	Referencia al dispositivo padre
Cardinalidad	0..1
Tipo	Reference (Device)
Notas	El recurso Reference, tiene campos que permiten indicar cualquier tipo de referencia, ya sea literal, relativa, interna o una URL absoluta. También puede indicar el tipo del recurso. Así como una referencia lógica cuando no se sabe la referencia literal. Incluso contiene un campo para indicar un texto alternativo en caso de que no haya ninguna referencia a otro recurso.
FIWARE Device	No se corresponde a nada en FIWARE. Está el campo controlledAsset pero no es exactamente lo mismo.

Tabla 83. FHIR: Device.parent

3.2.3 DeviceMetric – Descripción detallada

Un recurso DeviceMetric describe una capacidad de medida, cálculo o configuración de un dispositivo. Por un lado, describe propiedades estáticas que caracterizan la medida de una señal biológica o un cálculo producido por un dispositivo médico. Por otro, también puede ser usado para describir propiedades no estáticas, pero que son muy importantes para la medida como pueden ser el estado operacional, el modo de medida utilizado, la última fecha en la que fue calibrado, entre otros.

FHIR: DeviceMetric.identifier	
Definición	Son uno o varios identificadores únicos de instancia, asignados ya sea por una herramienta software, el fabricante, otras organizaciones o por los propietarios del dispositivo.
Cardinalidad	0..*
Tipo	Identifier
Notas	Es considerado como un identificador de negocio, no como un identificador lógico del recurso. Esto quiere decir que el identificador lógico del recurso cambia dependiendo del servidor FHIR al que sea enviado, o cambia si un recurso es movido de un lugar a otro. No obstante el identificador de negocio, independientemente del servidor al que sea enviado y de las copias que se hagan, siempre se refiere al mismo recurso, por lo que nos permite mantener la consistencia independientemente del contexto de uso. Todos los recursos FHIR que implementen este identifier, permiten ser buscados por este atributo en cualquier servidor.

FIWARE DeviceModel	Dado que las capacidades de medir un valor en FIWARE se especifican dentro del recurso DeviceModel, no hay nada equivalente a un id de un DeviceMetric. Se podría generar cogiendo como base el id del DeviceModel y añadiendo algo al final del id como por ejemplo el número que representa el orden de la capacidad en la lista de capacidades del dispositivo.

Tabla 84. FHIR: DeviceMetric.identifier

FHIR: DeviceMetric.type	
Definición	Tipo de la medida, ejemplos de medida son el pulso del corazón o el nivel de oxígeno en sangre.
Cardinalidad	1
Tipo	CodeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico. Los tipos se obtienen de los estándares LOINC [32] y IEEE 11073-10101 [33]
FIWARE DeviceModel	En DeviceModel encontramos el atributo obligatorio controlledProperty, su implementación es una lista de texto y contiene todo lo que el dispositivo puede medir, sentir o controlar. Cada una de sus entradas deberá corresponder con un recurso DeviceMetric asociado a ese dispositivo. En FHIR hay un catálogo más amplio de tipos que en FIWARE.

Tabla 85. FHIR: DeviceMetric.type

FHIR: DeviceMetric.unit	
Definición	Describe la unidad del valor observado del tipo especificado en type. Por ejemplo, porcentaje o segundos.
Cardinalidad	0..1
Tipo	CoadeableConcept
Notas	El tipo CodeableConcept de FHIR está formado por texto plano que representa el concepto y un elemento de tipo Coding que define el concepto dentro de un sistema terminológico.
FIWARE DeviceModel	En DeviceModel hay un campo opcional con el nombre de supportedUnits. Es una lista de texto, al igual que controlledProperty, y enumera las unidades de medidas soportadas por el dispositivo. En la documentación no se indica como utilizar la lista. No hay muchos ejemplos, así que se va a considerar que el orden de la lista de supportedUnits se refiere al elemento de controlledProperty en el mismo puesto.

Tabla 86. FHIR: DeviceMetric.unit

FHIR: DeviceMetric.source

Definición	Describe un enlace al recurso Device al que pertenece este DeviceMetric. Este recurso contiene la información administrativa del dispositivo como puede ser el fabricante y el número de serie.
Cardinalidad	0..1
Tipo	Reference (Device)
Notas	El recurso Reference, tiene campos que permiten indicar cualquier tipo de referencia, ya sea literal, relativa, interna o una URL absoluta. También puede indicar el tipo del recurso. Así como una referencia lógica cuando no se sabe la referencia literal. Incluso contiene un campo para indicar un texto alternativo en caso de que no haya ninguna referencia a otro recurso.
FIWARE DeviceModel	No se corresponde a nada en FIWARE.

Tabla 87. FHIR: DeviceMetric.source

FHIR: DeviceMetric.operationalStatus	
Definición	Indica el estado de operación actual del dispositivo. Por ejemplo, on, off, standby.
Cardinalidad	0..1
Tipo	code
Notas	
FIWARE Device	No se corresponde a nada en FIWARE. En Device de FIWARE hay un campo con el nombre de deviceState, que justo representa el estado del dispositivo desde el punto de vista operacional. No obstante, este campo ya se ha relacionado con un campo equivalente en Device. Se entiende que ese estado se refiere al estado operacional de una capacidad de medida concreta del dispositivo, pero el de FIWARE se refiere al dispositivo en general, por lo que la correspondencia que se ha realizado sigue siendo correcta. Este campo tiene un nivel superior de granularidad que no es reflejado en FIWARE, por lo que se ha decidido dejarlo sin mapear.

Tabla 88. FHIR: DeviceMetric.operationalStatus

FHIR: DeviceMetric.color	
Definición	Describe el color de representación para la medida. Esto se usa normalmente para ayudar a los médicos para hacer un seguimiento e identificar los tipos de parámetros por color.
Cardinalidad	0..1
Tipo	code
Notas	Un ejemplo práctico sería un dispositivo monitor asociado a un paciente que muestra los niveles de 3 constantes vitales, mostrando cada constante en un color diferente.
FIWARE	No se corresponde a nada en FIWARE.

Device	

Tabla 89. FHIR: DeviceMetric.color

FHIR: DeviceMetric.category	
Definición	Indica la categoría del proceso de generación de una observación médica. Un recurso DeviceMetric puede tomar los valores: setting, measurement, calculation o unspecified.
Cardinalidad	1
Tipo	code
Notas	
FIWARE DeviceModel	Este elemento a priori parece que se puede enlazar con el campo category de DeviceModel. Pero este es una lista con todas las categorías del dispositivo, que aparte que incluye varias que no son aplicables en este campo, al tener una lista no es posible saber cuál corresponde a cada capacidad de medida. Así que no se va a mapear.

Tabla 90. FHIR: DeviceMetric.category

FHIR: DeviceMetric.measurementPeriod	
Definición	Describe con qué frecuencia se realizan las medidas por parte del dispositivo. Puede variar desde milisegundos en un electrocardiograma hasta horas en otras pruebas.
Cardinalidad	0..1
Tipo	Timing
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 91. FHIR: DeviceMetric.measurementPeriod

FHIR: DeviceMetric.calibration	
Definición	Describe las calibraciones que han sido realizadas o que requieren realizarse.
Cardinalidad	0..*
Tipo	BackboneElement
Notas	El tipo BackboneElement usado en FHIR, indica que el elemento en cuestión tiene una serie de hijos que se definen dentro de él. A continuación, se procedrá a examinarlos uno a uno.

Tabla 92. FHIR: DeviceMetric.calibration

FHIR: DeviceMetric.calibration.type	
Definición	Describe el tipo del método de calibración. Los posibles valores son: unspecified, offset, gain y two-point.

Cardinalidad	0..1
Tipo	code
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 93. FHIR: DeviceMetric.calibration.type

FHIR: DeviceMetric.calibration.state	
Definición	Describe el estado de calibración. Los posibles valores son: not-calibrated, calibration-required,calibrated y unspecified.
Cardinalidad	0..1
Tipo	code
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 94. FHIR: DeviceMetric.calibration.state

FHIR: DeviceMetric.calibration.time	
Definición	Describe cuando se realizó la última calibración
Cardinalidad	0..1
Tipo	instant
FIWARE Device	Hay una relación directa con el campo de FIWARE dateLastCalibration, tiene exactamente la misma función.

Tabla 95. FHIR: DeviceMetric.calibration.time

3.2.4 Observation – Descripción detallada

Un recurso de tipo Observation contiene medidas y afirmaciones simples hechas sobre un paciente, dispositivo u otro sujeto. Las observaciones son un elemento central en el sistema médico. La inmensa mayoría de las observaciones son algunos pares de atributos nombre/valor con algo de metadatos, pero también existen observaciones más complejas que agrupan otras observaciones, e incluso observaciones con varios componentes.

FHIR: Observation.identifier	
Definición	Identificador único asignado a esta observación.
Cardinalidad	0..*
Tipo	Identifier
Notas	Es considerado como un identificador de negocio, no como un identificador lógico del recurso. El identificador lógico del recurso cambia dependiendo del servidor FHIR al que sea enviado, o cambia si un recurso es movido de un lugar

	a otro. No obstante, el identificador de negocio, independientemente del servidor al que sea enviado y de las copias que se hagan, siempre se refiere al mismo recurso, por lo que nos permite mantener la consistencia independientemente del contexto de uso. Todos los recursos FHIR que implementen este identifier, permiten ser buscados por este atributo en cualquier servidor.
FIWARE Device	Dado que el value en FIWARE está dentro del recurso Device. En FIWARE no tiene un identificador propio. Teniendo en cuenta que un Device normalmente va a generar más de una observación, una solución sería generar un id cogiendo el identificar del Device como base y añadiendo al final el número de la observación generada por ese Device.

Tabla 96. FHIR: Observation.identifier

FHIR: Observation.basedOn	
Definición	Un plan, propuesta o petición que es completada total o parcialmente por este evento.
Cardinalidad	0..*
Tipo	Reference (CarePlan | DeviceRequest | ImmunizationRecommendation | MedicationRequest | NutritionOrder | ServiceRequest)
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 97. FHIR: Observation.basedOn

FHIR: Observation.partOf	
Definición	Un evento más grande, del cual este recurso es un componente o paso.
Cardinalidad	0..*
Tipo	Reference (MedicationAdministration | MedicationDispense | MedicationStatement | Procedure | Immunization | ImagingStudy)
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 98. FHIR: Observation.partOf

FHIR: Observation.status	
Definición	El estado del valor resultante. Puede tomar los valores: registered | preliminary | final | amended
Cardinalidad	1
Tipo	code
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 99. FHIR: Observation.status

FHIR: Observation.category	
Definición	Un código que clasifica el tipo general de observación que se está haciendo.
Cardinalidad	0..*
Tipo	CodeableConcept
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 100. FHIR: Observation.category

FHIR: Observation.code	
Definición	Describe que fue observado. A veces también se llama el "nombre" de la observación.
Cardinalidad	1
Tipo	CodeableConcept
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 101. FHIR: Observation.code

FHIR: Observation.subject	
Definición	El paciente, o grupo de pacientes, localización o dispositivo del que trata esta observación y donde se coloca la observación. Si el objetivo de la observación es diferente del sujeto, los campos focus o el propio campo code especifican el objetivo real de la observación.
Cardinalidad	0..1
Tipo	Reference (Patient | Group | Device | Location)
FIWARE Device	No se corresponde a nada en FIWARE. Ya que la observación del valor tiene lugar dentro del Device en FIWARE.

Tabla 102. FHIR: Observation.subject

FHIR: Observation.focus	
Definición	Este campo sólo se usa en ocasiones especiales. Normalmente basta con los campos code y subject. Este campo se utiliza en el caso de que el objetivo no sea el paciente en cuestión, sino algo o alguien asociado al paciente ya sea su pareja padre, feto o donante.
Cardinalidad	0..*
Tipo	Reference (Any)
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 103. FHIR: Observation.focus

FHIR: Observation.encounter	
Definición	El evento médico durante el cual se realiza la observación.
Cardinalidad	0..1
Tipo	Reference (Encounter)
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 104. FHIR: Observation.encounter

FHIR: Observation.effective[X]	
Definición	El instante o periodo de tiempo en el que se hizo la observación.
Cardinalidad	0..1
Tipo	Choice [datetime | Period | Timing | Instant]
Notas	Al menos una fecha debería estar presente, a menos que la observación sea un reporte histórico.
FIWARE Device	Es una correspondencia directa con el medatado timestamp del campo value de Device.

Tabla 105. FHIR: Observation.effective[X]

FHIR: Observation.issued	
Definición	La fecha y hora en la que hicieron disponible esta versión de la observación, normalmente después de que los resultados hayan sido revisados y verificados.
Cardinalidad	0..1
Tipo	instant
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 106. FHIR: Observation.issued

FHIR: Observation.performer	
Definición	Indica quién fue responsable de indicar que el valor de la observación es verdadero
Cardinalidad	0..*
Tipo	Reference (Practitioner | PractitionerRole | Organization | CareTeam | Patient | RelatedPerson)
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 107. FHIR: Observation.performer

FHIR: Observation.value[X]	
Definición	La información resultante de hacer la observación, en el caso de que sea un valor simple.
Cardinalidad	0..1
Tipo	Choice [Quantity|CodeableConcept|string|boolean|integer|Range|Ratio| SampledData|time|dateTime|Period]
FIWARE Device	Hay una correspondencia directa con el campo value de Device.

Tabla 108. FHIR: Observation.value[X]

FHIR: Observation.dateAbsentReason	
Definición	Proporciona la razón de por qué el valor esperado en el campo value de este recurso no existe, si fuese necesario.
Cardinalidad	0..1
Tipo	CodeableConcept
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 109. FHIR: Observation.dateAbsentReason

FHIR: Observation.interpretation	
Definición	Una valoración categórica de un valor observado, por ejemplo, alto, bajo o normal.
Cardinalidad	0..*
Tipo	CodeableConcept
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 110. FHIR: Observation.interpretation

FHIR: Observation.note	
Definición	Comentarios sobre la observación de los resultados
Cardinalidad	0..*
Tipo	Annotation
Notas	Puede incluir afirmaciones generales sobre la observación, o afirmaciones sobre valores significantes, inesperados, o poco fiables. También puede haber comentarios sobre la fuente de la información si fuese relevante.
FIWARE	No se corresponde a nada en FIWARE.

Device	

Tabla 111. FHIR: Observation.note

FHIR: Observation.bodySite	
Definición	Indica en qué parte del cuerpo del sujeto fue realizada la observación.
Cardinalidad	0..1
Tipo	CodeableConcept
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 112. FHIR: Observation.bodySite

FHIR: Observation.method	
Definición	Indica el mecanismo usado para realizar la observación.
Cardinalidad	0..1
Tipo	CodeableConcept
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 113. FHIR: Observation.method

FHIR: Observation.specimen	
Definición	Referencia al recurso de tipo Specimen que fue utilizado para hacer esta observación. Este recurso modela una muestra o material tomado de una entidad biológica viva o muerta o de un objeto físico del ambiente.
Cardinalidad	0..1
Tipo	Reference (Specimen)
FIWARE Device	No se corresponde a nada en FIWARE.

Tabla 114. FHIR: Observation.specimen

FHIR: Observation.device	
Definición	Referencia al dispositivo usado para generar los datos de esta observación.
Cardinalidad	0..1
Tipo	Reference (Device | DeviceMetric)
FIWARE Device	No se corresponde a nada en FIWARE. Ya que el valor medido aparece dentro del Device en sí.

Tabla 115. FHIR: Observation.device

FHIR: Observation.referenceRange	
Definición	Guía de cómo interpretar el valor de la observación, en comparación a un rango de valores normal o recomendado. Si hay más de una referencia se interpretan como un OR lógico. En otras palabras, para representar dos poblaciones distintas de objetivos, se usarían dos elementos referenceRange.
Cardinalidad	0..*
Tipo	BackboneElement
FIWARE Device	No se corresponden a nada en FIWARE ninguno de sus elementos.

Tabla 116. FHIR: Observation.referenceRange

FHIR: Observation.referenceRange.low	
Definición	El valor del límite inferior del rango de referencia. El valor límite se incluye en el rango, en caso de ser omitido, se considera que es insignificante.
Cardinalidad	0..1
Tipo	SimpleQuantity

Tabla 117. FHIR: Observation.referenceRange.low

FHIR: Observation.referenceRange.high	
Definición	El valor del límite superior del valor de referencia. El valor límite se incluye en el rango, en caso de ser omitido se considera que es insignificante.
Cardinalidad	0..1
Tipo	SimpleQuantity

Tabla 118. FHIR: Observation.referenceRange.high

FHIR: Observation.referenceRange.type	
Definición	En caso de haber más de un rango de valores, este código indica cual debe aplicarse al valor medido. En caso de omitirse este campo, se considera el rango normal de valores.
Cardinalidad	SimpleQuantity
Tipo	0..1

Tabla 119. FHIR: Observation.referenceRange.type

FHIR: Observation.referenceRange.appliesTo	
Definición	Códigos para indicar a qué tipo de población objetivo aplica este rango. Por ejemplo, un rango de referencia se puede basar en la población normal, o en sexo o raza concretos. Al haber varios elementos appliesTo, son interpretados como un AND lógico.
Cardinalidad	0..*

Tipo	CodeableConcept

Tabla 120. FHIR: Observation.referenceRange.appliesTo

FHIR: Observation.referenceRange.age	
Definición	La edad en la cual este rango es aplicable.
Cardinalidad	0..1
Tipo	Range

Tabla 121. FHIR: Observation.referenceRange.age

FHIR: Observation.referenceRange.text	
Definición	Rango de referencia basado en texto de una observación, puede ser usado cuando un rango cuantitativo no es apropiado para la observación. Por ejemplo, un valor de referencia “Negativo”
Cardinalidad	0..1
Tipo	string

Tabla 122. FHIR: Observation.referenceRange.text

FHIR: Observation.hasMember	
Definición	Recurso relacionado que pertenece al grupo de la observación.
Cardinalidad	0..*
Tipo	Reference (Observation | QuestionnaireResponse | MolecularSequence)
FIWARE Device	No se corresponde a nada en FIWARE

Tabla 123. FHIR: Observation.hasMember

FHIR: Observation.derivedFrom	
Definición	Medidas relacionadas de las que está compuesta la observación.
Cardinalidad	0..*
Tipo	Reference (DocumentReference | ImagingStudy | Media | QuestionnaireResponse | Observation | MolecularSequence)
FIWARE Device	No se corresponde a nada en FIWARE

Tabla 124. FHIR: Observation.derivedFrom

FHIR: Observation.component	
Definición	Algunas observaciones tienen múltiples componentes de la observación. Cada uno de estos componentes son expresados como valores separados de pares código-valor que comparten los mismos atributos.

Cardinalidad	0..*
Tipo	BackboneElement
Notas	Cada uno de los componentes de la observación compleja comparten los mismos atributos de la Observación primaria y siempre son tratados en conjunto como una única observación. No obstante, el rango de referencia de la observación primario no es heredado por los componentes y es necesario indicar un rango para cada uno de ellos.
FIWARE Device	No es aplicable a FIWARE ya que no permite, por defecto al menos, tomar valores tan complejos que necesiten este campo.

Tabla 125. FHIR: Observation.component

FHIR: Observation.component.code	
Definición	Describe que fue observado.
Cardinalidad	1..1
Tipo	CodeableConcept

Tabla 126. FHIR: Observation.component.code

FHIR: Observation.component.value[X]	
Definición	La información determinada como resultado de hacer la observación, si la información fuese un valor simple.
Cardinalidad	0..1
Tipo	Choice[Quantity|CodeableConcept|string|boolean|integer|Range|Ratio|SampledData|time|dateTime|Period]

Tabla 127. FHIR: Observation.component.value[X]

FHIR: Observation.component.dataAbsentReason	
Definición	Proporciona una razón de por qué el valor esperado en el elemento Observation.component.value[] no está disponible.
Cardinalidad	0..1
Tipo	CodeableConcept

Tabla 128. FHIR: Observation.component.dataAbsentReason

FHIR: Observation.component.interpretation	
Definición	Una afirmación categórica de un valor observado. Por ejemplo, alto, bajo o normal.
Cardinalidad	0..*
Tipo	CodeableConcept

Tabla 129. FHIR: Observation.component.interpretation

FHIR: Observation.component.referenceRange	
Definición	Proporciona una guía de cómo interpretar el valor en comparación a un rango normal o recomendado.
Cardinalidad	0..*
Tipo	BackboneElement
Notas	Ver Observationt.referenceRange para ver la implementación completa.

Tabla 130. FHIR: Observation.component.referenceRange

3.2.5 Resumen de la correspondencia

FHIR		FIWARE	
RECURSO	CAMPO	RECURSO	CAMPO
DeviceDefinition	identifier[0].value	DeviceModel	id
DeviceDefinition	identifier[0].assigner.reference	DeviceModel	source
DeviceDefinition	deviceName.type == model-name deviceName.value	DeviceModel	modelName
DeviceDefinition	deviceName.type == user-friendly-name deviceName.name	DeviceModel	name
DeviceDefinition	deviceName.type == manufacturer-name deviceName.name	DeviceModel	manufacturerName
DeviceDefinition	deviceName.type == other deviceName.name	DeviceModel	brandName
DeviceDefinition	type.display	DeviceModel	category
DeviceDefinition	Por cada specialization.systemType	DeviceModel	Añadir un elemento a la lista supportedProtocol
DeviceDefinition	Por cada capability.type.text	DeviceModel	Añadir un elemento a la lista function
DeviceDefinition	contact[0].value	DeviceModel	dataProvider
DeviceDefinition	onlineInformation	DeviceModel	documentation
DeviceDefinition	note[0].text	DeviceModel	description

Device	identifier[0].value	Device	id
Device	definition	Device	refDeviceModel
Device	status	Device	deviceState (Es uno de los 5 valores que actualiza la aplicación)
Device	manufacturer	Device	provider
Device	manufacturerDate	Device	dateManufactured
Device	serialNumber	Device	serialNumber
Device	deviceName.type == UserFriendlyName deviceName.name	Device	name
Device	type	Device	category
Device	Por cada specialization.systemType.text	Device	Añadir un elemento a la lista supportedProtocol
Device	version.type == hardware versión.value	Device	hardwareVersion
Device	version.type == software versión.value	Device	softwareVersion
Device	version.type == firmware versión.value	Device	firmwareVersion
Device	version.type == os versión.value	Device	osVersion
Device	property.type property.value	Device	Añadir elemento al mapa configuration {type,value}
Device	patient.reference	Device	Añadir elemento a la lista owner
Device	owner.reference	Device	Añadir elemento a la lista owner
Device	contact[0].value	Device	dataProvider
Device	location	Device	location (Es uno de los 5 valores que actualiza la aplicación)
Device	url	Device	ipAddress (Es uno de los 5 valores que actualiza la aplicación)
Device	note[0].text	Device	description
DeviceMetric	type.text	DeviceModel	Añadir elemento a la lista controlledProperty
DeviceMetric	unit[0].display	DeviceModel	Añadir elemento a la lista supportedUnits
DeviceMetric	calibration.time	Device	dateLastCalibration

			(Es uno de los 5 valores que actualiza la aplicación)
Observation	effective[X]	Device	value.timestamp
Observation	value[X]	Device	value (Es uno de los 5 valores que actualiza la aplicación)

Tabla 131. Resumen correspondencia FHIR-FIWARE

3.3 Modelo de datos

3.3.1 Vista general

Este diagrama, de forma resumida, muestra todas las clases e interfaces que se han desarrollado para este TFG. Toda la funcionalidad de las interfaces está definida en la clase Controller. Este controlador, dependiendo de la función que tiene que llevar a cabo, usa de 1 a 3 clientes diferentes.

El cliente FIWARE (clase FiwClient) permite la comunicación con el Orion Context Broker. Maneja suscripciones, entidades, y objetos DeviceFiw y DeviceModel que modelan los recursos Device y DeviceMode de FIWARE respectivamente. El controlador maneja la base de datos a través de los métodos definidos en las interfaces TemplateDBRepository y TemplateDeviceRepository. Hay dos tipos de entradas de la base de datos, que se definen mediante las clases TemplateDB y TemplateDevice. Finalmente, en la interfaz FhirClientInterface se definen los métodos para manejar recursos en un servidor FHIR.

La clase auxiliar NgsiParser se utiliza para convertir al estándar NGSIv2 la información contenida en objetos de tipo DeviceFiw y DeviceModel. La clase FhirToFiware realiza la conversión de recursos FHIR a recursos FIWARE según la correspondencia establecida en el apartado anterior. La clase DeviceUpdate modela una notificación del Context Broker.

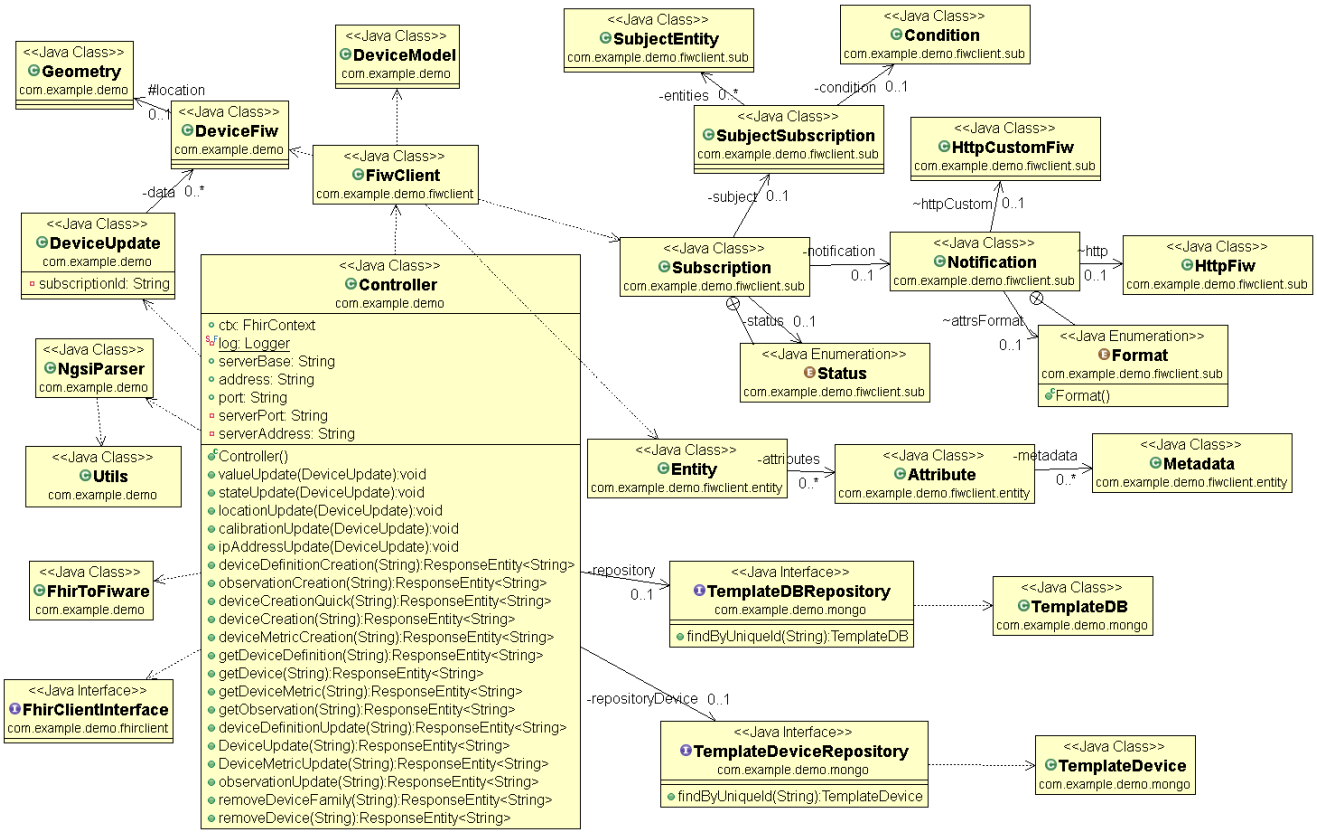

Figura 45. Diagrama de clases general

3.3.2 Modelo del recurso FIWARE Device

En el repositorio de FIWARE en Github [34] está el recurso modelado en Python, pero no se ha encontrado nada en Java. Así que se ha creado esta clase a partir de la especificación del modelo de datos.

Figura 46. Diagrama de clases Device FIWARE

3.3.3 Modelo del recurso FIWARE DeviceModel

En el repositorio de FIWARE en Github [34] está el recurso modelado en Python, pero no se ha encontrado nada en Java. Así que se ha creado esta clase a partir de la especificación del modelo de datos.

Figura 47. Diagrama de clase DeviceModel

3.3.4 Modelo de la posición según GeoJSON

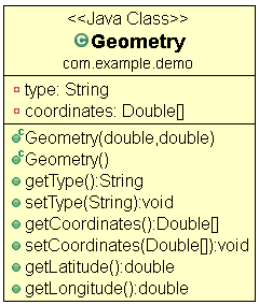

Figura 48. Diagrama de clase GeoJSON

3.3.5 Conversor al estándar NGSIv2

Mediante los métodos estáticos de esta clase, se puede dar el formato que impone el estándar NGSIv2 a los recursos DeviceFiw y DeviceModel. Como se ha comentado con anterioridad, el Orion Context Broker trabaja con el formato NGSIv2 por lo que este paso intermedio es necesario para poder, por ejemplo, crear los recursos en Orion desde la aplicación Java.

La otra opción hubiera sido modelar las clases DeviceFiw y DeviceModel directamente según NGSIv2. No obstante, hubiera complicado mucho el modelo de datos y se pensó que era mejor crear este conversor intermedio.

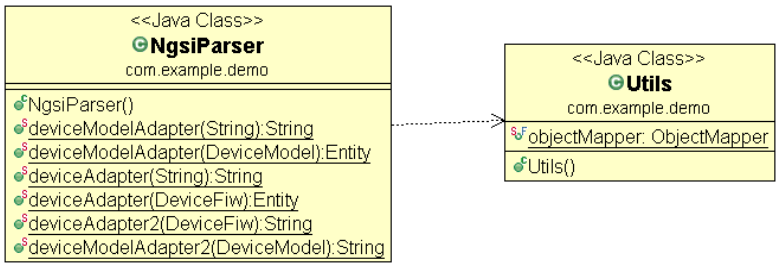

Figura 49. Diagrama de clases NgsiParser

3.3.6 Conversor FHIR-FIWARE

Esta clase es la materialización del estudio comparativo y la correspondencia de los recursos del mundo FHIR al mundo FIWARE. Presenta métodos estáticos para pasar de los recursos de FHIR a los recursos FIWARE.

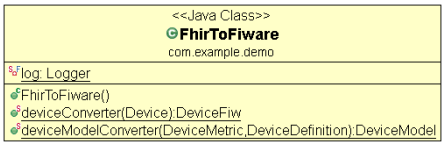

Figura 50. Diagrama de clase FhirToFiware

3.3.7 Modelo de las entidades NGSIv2

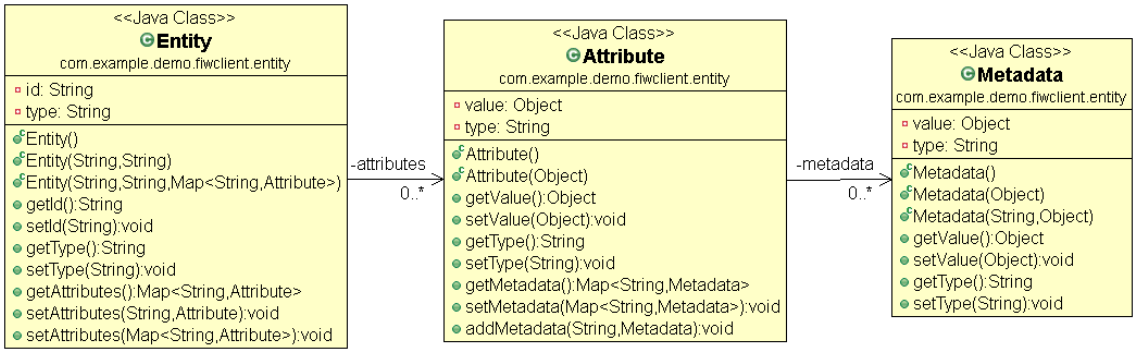

Figura 51. Diagrama de clases Entity NGSIv2

3.3.8 Modelo de las suscripciones NGSIv2

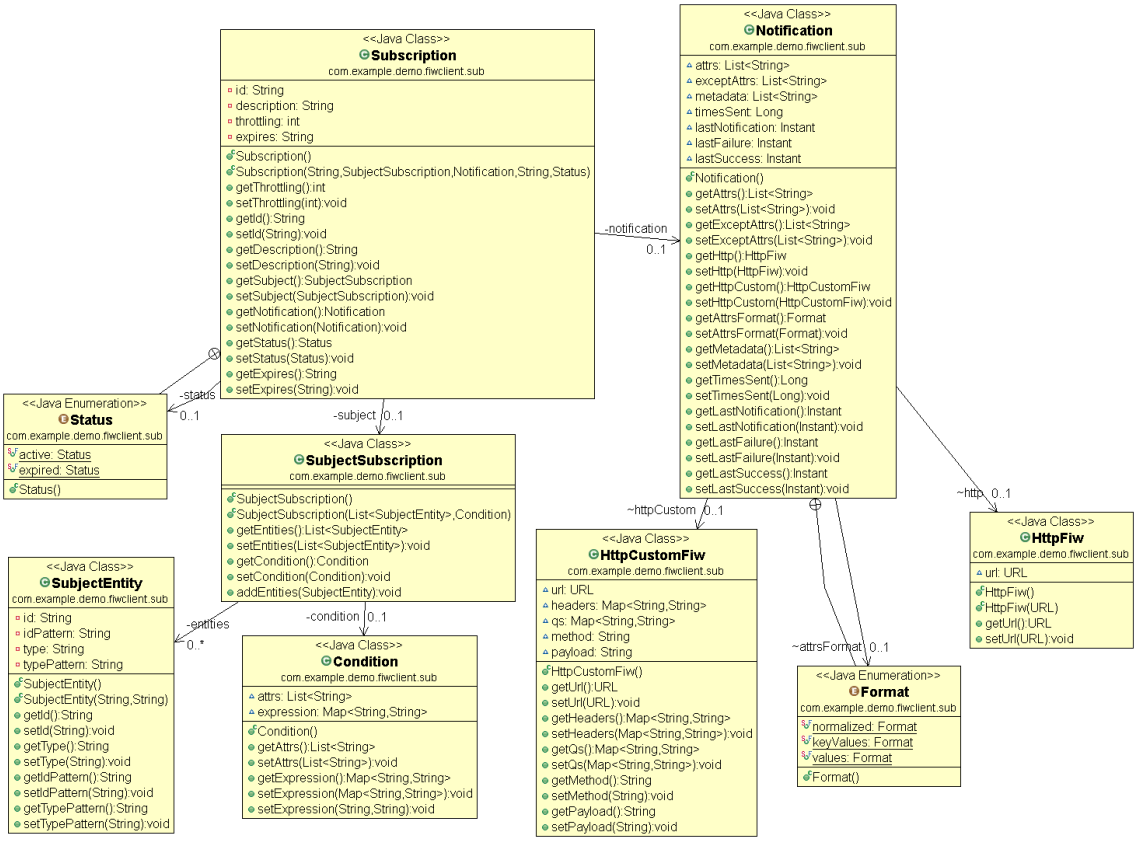

Figura 52. Diagrama de clases Subscription

3.3.9 Modelo de las notificaciones de Orion

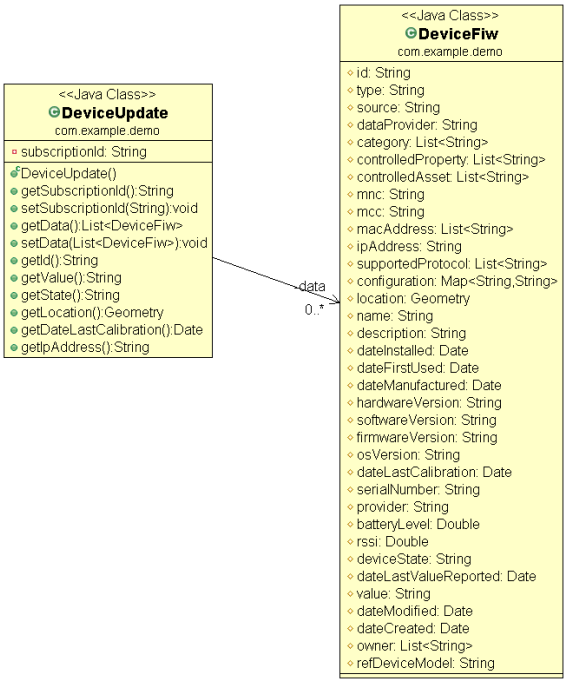

Figura 53. Diagrama de clases DeviceUpdate

3.4 Asignación de Identificadores

3.4.1 Identificadores de los recursos FHIR-FIWARE

En primer lugar, hay que asignar un identificador que englobe a toda una familia de dispositivos, tanto en FHIR como en FIWARE. Esto afecta a los recursos DeviceModel (FIWARE) y DeviceDefinition (FHIR). Se ha decidido usar el mismo identificador para ambos y se construye así:

"fiware_{nombre del tipo de dispositivo}"

Una vez que se cuenta con un identificador para el conjunto de dispositivos, hay que asignar un identificador a cada uno de los dispositivos físicos que forman parte de la familia. Esto afecta a los recursos Device de ambas plataformas. Se ha decidido usar el mismo identificador para ambos y se construye así:

"fiware_{nombre del tipo de dispositivo}_{número asignado al dispositivo físico}"

En FIWARE sólo se usan los recursos DeviceModel y Device. No obstante, en FHIR también se manejan recursos DeviceMetric y Observation. Un dispositivo puede tener más de una capacidad de medida (DeviceMetric) y obviamente puede generar cualquier número de observaciones. Se ha decidido nombrar a los recursos DeviceMetric así:

"fiware_{nombre del tipo de dispositivo}_{número asignado al dispositivo físico}_{número asignado a la capacidad}"

La primera capacidad tendrá el número 1 y así incrementalmente. En cuanto a las observaciones se emplea algo similar:

"fiware_{nombre del tipo de dispositivo}_{número asignado al dispositivo físico}_{número de la observación}"

La primera observación tendrá el número 1 y así incrementalmente. Como ejemplo, la familia de glucómetros del apartado de pruebas tiene los siguientes identificadores:

- Los recursos DeviceModel y DeviceDefinition tienen el identificador **"fiware_glucometer"**
- Se ha creado 3 dispositivos dentro de la familia con los identificadores:
  - **"fiware_glucometer_1"**
  - **"fiware_glucometer_2"**
  - **"fiware_glucometer_3"**
- En este caso los dispositivos sólo tienen una capacidad de medida y se identifica así:
  - **"fiware_glucometer_1_1"**
  - **"fiware_glucometer_2_1"**
  - **"fiware_glucometer_3_1"**
- Si se supone que el primer dispositivo ha realizado 5 observaciones, el segundo 10 y el tercero 15, tendremos las observaciones con los siguientes identificadores:
  - de **"fiware_glucometer_1_1"** hasta **"fiware_glucometer_1_5"**
  - de **"fiware_glucometer_2_1"** hasta **"fiware_glucometer_2_10"**
  - de **"fiware_glucometer_3_1"** hasta **"fiware_glucometer_3_15"**

3.4.2 Identificadores de la base de datos

En la base de datos MongoDB, se crea una entrada por cada familia de dispositivos y una entrada por cada dispositivo perteneciente a la familia. En el ejemplo de arriba, se habrán creado por tanto 4 entradas en la base de datos.

La entrada correspondiente a la familia de dispositivos se identifica con el mismo identificador que los recursos DeviceModel y DeviceDefinition, esto es:

"fiware_{nombre del tipo de dispositivo}"

Para cada una de las entradas asociadas a los dispositivos físicos, se usa el mismo identificador que en los recursos Device de ambas plataformas, esto es:

"fiware_{nombre del tipo de dispositivo}_{número asignado al dispositivo físico}"

Por tanto, siguiendo el mismo ejemplo se tendrían estos identificadores:

- Entrada asociada a la familia de dispositivos:
  - "**fiware_glucometer"**
- **Entradas asociadas a cada uno de los dispositivos físicos:**
  - **"fiware_glucometer_1"**
  - **"fiware_glucometer_2"**
  - **"fiware_glucometer_3"**

3.5 Cliente FHIR

La librería HAPI FHIR usada en este proyecto ofrece dos tipos de clientes RESTful para interaccionar con servidores FHIR. La primera opción es un cliente fluido/genérico y la segunda un cliente basado en anotaciones.

El cliente genérico es más fácil de usar y generalmente proporciona una forma más rápida de empezar. Sin embargo, el cliente basado en anotaciones confía en una vinculación estática a una serie de operaciones específicas, lo que da una mejor comprobación de errores en tiempo de compilación. Este segundo modelo requiere más esfuerzo para usarlo, pero puede ser muy útil si la persona que especifica los métodos a invocar no es la misma que los usa. Dado que está pensado reutilizar este trabajo, se ha optado por el cliente basado en anotaciones.

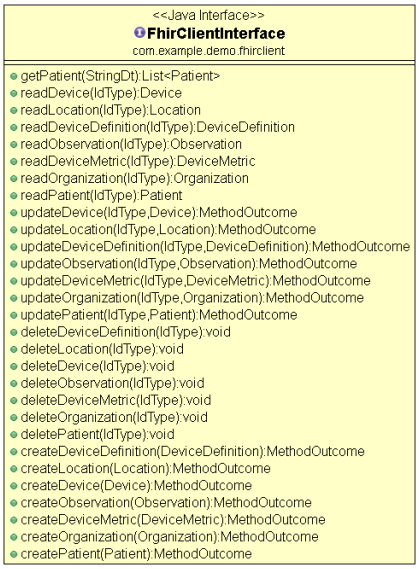

Figura 54. Diagrama de clase FhirClientInterface

3.5.1 Definición de una interfaz cliente RESTful

El primer paso para crear un cliente basado en anotaciones consiste en definir una interfaz RESTful.

Una clase que describa una interfaz de cliente RESTful debe extender a la interfaz IRestfulClient, y contendrá uno o más métodos que habrán sido anotados con anotaciones especiales que indican que operación RESTful soportan.

Un ejemplo sería:

```
public interface FhirClientInterface extends IRestfulClient{

    //example
    @Search
    public List<Patient> getPatient(@RequiredParam(name = Patient.SP_FAMILY) StringDt theFamilyName);
```

Figura 55. Ejemplo de interfaz FhirClientInterface

A continuación, se van a describir los métodos anotados que se han usado en la aplicación.

3.5.1.1 Métodos de la interfaz FhirClientInterface para obtener recursos del servidor

```
//obtener Device
@Read
Device readDevice(@IdParam IdType theId);

//obtener Location
@Read
Location readLocation(@IdParam IdType theId);

//obtener DeviceDefinition
@Read
DeviceDefinition readDeviceDefinition(@IdParam IdType theId);

//obtener Observation
@Read
Observation readObservation(@IdParam IdType theId);

//obtener DeviceMetric
@Read
DeviceMetric readDeviceMetric(@IdParam IdType theId);

//obtener Organization
@Read
Organization readOrganization(@IdParam IdType theId);

//obtener Patient
@Read
Patient readPatient(@IdParam IdType theId);
```

Figura 56. Interfaz FhirClientInterface@Read

3.5.1.2 Métodos de la interfaz FhirClientInterface para actualizar recursos del servidor

```
//actualizar Device
@Update
public abstract MethodOutcome updateDevice(@IdParam IdType theId, @ResourceParam Device theDevice);

//actualizar Location
@Update
public abstract MethodOutcome updateLocation(@IdParam IdType theId, @ResourceParam Location theLocation);

//actualizar DeviceDefinition
@Update
public abstract MethodOutcome updateDeviceDefinition(@IdParam IdType theId, @ResourceParam DeviceDefinition theDeviceDefinition);

//actualizar Observation
@Update
public abstract MethodOutcome updateObservation(@IdParam IdType theId, @ResourceParam Observation theObservation);

//actualizar DeviceMetric
@Update
public abstract MethodOutcome updateDeviceMetric(@IdParam IdType theId, @ResourceParam DeviceMetric theDeviceMatric);

//actualizar Organization
@Update
public abstract MethodOutcome updateOrganization(@IdParam IdType theId, @ResourceParam Organization theOrganization);

//actualizar Patient
@Update
public abstract MethodOutcome updatePatient(@IdParam IdType theId, @ResourceParam Patient thePatient);
```

Figura 57. Interfaz FhirClientInterface@Update

3.5.1.3 Métodos de la interfaz FhirClientInterface para borrar recursos del servidor

```
//borrar DeviceDefinition
@Delete(type=DeviceDefinition.class)
public void deleteDeviceDefinition(@IdParam IdType theId);

//borrar Location
@Delete(type=Location.class)
public void deleteLocation(@IdParam IdType theId);

//borrar Device
@Delete(type=Device.class)
public void deleteDevice(@IdParam IdType theId);

//borrar Observation
@Delete(type=Observation.class)
public void deleteObservation(@IdParam IdType theId);

//borrar DeviceMetric
@Delete(type=DeviceMetric.class)
public void deleteDeviceMetric(@IdParam IdType theId);

//borrar Organization
@Delete(type=Organization.class)
public void deleteOrganization(@IdParam IdType theId);

//borrar Patient
@Delete(type=Patient.class)
public void deletePatient(@IdParam IdType theId);
```

Figura 58. Interfaz FhirClientInterface@Delete

3.5.1.4 Métodos de la interfaz FhirClientInterface para crear recursos del servidor

```
//crear DeviceDefinition
@Create
public abstract MethodOutcome createDeviceDefinition(@ResourceParam DeviceDefinition theDeviceDefinition);

//crear DeviceDefinition
@Create
public abstract MethodOutcome createLocation(@ResourceParam Location thelocation);

//crear Device
@Create
public abstract MethodOutcome createDevice(@ResourceParam Device theDevice);

//crear Observation
@Create
public abstract MethodOutcome createObservation(@ResourceParam Observation theObservation);

//crear DeviceMetric
@Create
public abstract MethodOutcome createDeviceMetric(@ResourceParam DeviceMetric theDeviceMetric);

//crear Organization
@Create
public abstract MethodOutcome createOrganization(@ResourceParam Organization theOrganization);

//crear Patient
@Create
public abstract MethodOutcome createPatient(@ResourceParam Patient thePatient);
```

Figura 59. Interfaz FhirClientInterface@Create.

3.5.2 Instanciación del cliente RESTful

Una vez que la interfaz del cliente haya sido definida, hay que crear un objeto FhirContext e instanciar el cliente que estaría listo para usarse. La librería HAPI FHIR se encarga de la implementación de los métodos de la interfaz. La creación del objeto FhirContext es bastante costosa, por lo que hay que controlar el número de objetos FhirContext que se crean en la vida de la aplicación. En la pasarela, sólo se crea una única vez al lanzar la aplicación.

```java
public static void main(String[] args) {
    FhirContext ctx = FhirContext.forR4();
    String serverBase = "http://foo.com/fhirServerBase";

    //Creación del cliente
    IRestfulClient client = ctx.newRestfulClient(IRestfulClient.class, serverBase);

    //Prueba del client
    List<Patient> patients = client.getPatient(new StringDt("SMITH"));

}
```

Figura 60. Instanciación cliente FHIR

3.5.3 Configurar la codificación (JSON/XML)

La interfaz IRestfulClient, que hemos extendido a la hora de crear la interfaz del cliente, viene con algunos métodos muy útiles para configurar la interacción con el servidor.

En la siguiente imagen, se muestra como configurar el cliente para elegir entre la codificación JSON o XML. Además, se muestra como pedir que las respuestas sean "pretty printed", es decir, que el texto de la respuesta venga espaciado de una forma que sea más fácil de leer para los humanos.

```java
//Creación del cliente
FhirContext ctx = FhirContext.forR4();
IPatientClient client = ctx.newRestfulClient(IPatientClient.class, "http://localhost:8080");

// Codificación JSON en las peticiones
client.setEncoding(EncodingEnum.JSON);

// Respuestas "pretty printed"
client.setPrettyPrint(true);
```

Figura 61. Configuración cliente FHIR

3.6 Cliente FIWARE

Tras investigar por la red, no se encontró ningún cliente NGSIv2 completo, actualizado y programado en java. Por lo tanto, se decidió crear uno con ayuda de la clase RestTemplate de la librería Spring usada en este proyecto. RestTemplate es la clase central de Spring para programar aplicaciones cliente HTTP y consumir servicios web Restful. Proporciona métodos para consumir servicios web para todos los distintos tipos de mensajes HTTP, ya sea GET, POST, PUT, DELETE, etc.

Además del cliente, se ha modelado en java los recursos entidad y suscripción según el estándar NGSIv2. Están especificados en el apartado dedicado al modelado de la aplicación.

3.6.1 Suscripciones y Notificaciones

En el apartado dedicado a NGSIv2 dentro del capítulo 2, se especificó la estructura tanto de las suscripciones como de las notificaciones dentro de este estándar. Se observó entonces que las suscripciones permiten especificar filtros con la granularidad que se desee. Además, permiten especificar completamente el contenido del mensaje de notificación.

En este proyecto, cuando el gestor crea una nueva familia de dispositivos, se crean 5 nuevas suscripciones dentro del Orion Context Broker. Estas suscripciones notifican cambios en 5 atributos clave de las entidades tipo Device pertenecientes a esta nueva familia. Los atributos, como se han comentado con anterioridad, son: "value", "deviceState", "dateLastCalibration", "location" e "ipAddress". Estos campos son los que se modifican en el funcionamiento del

dispositivo, a diferencia del resto que presentan un carácter más estático. Las notificaciones se realizan mediante mensajes HTTP POST, la aplicación presenta una interfaz que captura estas notificaciones asíncronas y las procesa.

Las 5 suscripciones son bastante similares entre sí, se va a analizar una de las suscripciones del apartado de pruebas con los glucómetros.


```
{
    "id": "5d83bbb0996a4c7cc9104c1e",
    "description": "Suscripción a cambios en la localización del sensor",
    "expires": "2022-01-01T14:00:00.00Z",
    "status": "active",
    "subject": {
        "entities": [
            {
                "idPattern": "^fiware_glucometer",
                "type": "Device"
            }
        ],
        "condition": {
            "attrs": [
                "location"
            ]
        }
    },
    "notification": {
        "attrs": [
            "id",
            "location"
        ],
        "attrsFormat": "keyValues",
        "http": {
            "url": "http://192.168.108.129:9090/location"
        }
    },
    "throttling": 1
},
```


Figura 62. Ejemplo de suscripción

Campos de la suscripción a cambios en la localización del dispositivo:

- **“id”:** es un identificador único que asigna automáticamente el Orion Context Broker al enviar la suscripción.
- **“description”:** es un breve recordatorio del objetivo de la suscripción.
- **“expires”:** es la fecha en la que caducará la suscripción.
- **“status”:** indica que la suscripción está activa, este atributo ha sido creado por Orion de forma automática.
- **“subject”:** mediante este campo se especifica qué desencadenará la suscripción.
    - **“entities”:** se filtran todas las entidades de tipo Device cuyo identificador comienze por **“fiware_glucometer”**. Así se selecciona la familia entera de dispositivos.
    - **“condition”:** este campo es opcional, aquí se especifica que se desencadenará la notificación cuando cambie el atributo **“location”.** Por lo tanto, el filtro hace que salte la notificación cuando cambie el atributo **“location”,** de las entidades de tipo Device que comiencen por **“fiware_glucometer”.**
- **“notification”:** describe el mensaje de notificación.
    - **“attrs”:** lista de atributos que contendrá la notificación, en este caso sólo interesa mandar el identificador del recurso que ha sido modificado y el nuevo valor del campo **“location”**. Con el identificador, la aplicación recupera toda la información estática asociada de la base de datos y la completa con el nuevo valor medido.

    - **“attrsFormat”:** indica el modo de representación de la entidad, al seleccionar “keyValues” solo se envían los valores de los atributos. No se indican por tanto ni su tipo ni los metadatos asociados.
    - **“http”**: dirección y puerto al que se enviará la notificación. La dirección IP es la de la máquina que ejecuta la aplicación java. Al ejecutar el programa, el gestor indica por la línea de comandos, entre otras cosas, la dirección IP del ordenador. Así, al generar las suscripciones se hacen con la IP correcta. El puerto 9090, es el puerto por defecto de la aplicación, también se puede cambiar por la línea de comandos. En este caso, se envía a la dirección “{base}/location”. Este es el endpoint que procesa la notificación de cambio de localización.

- **“throttling”:** indica el periodo mínimo en segundos que deben pasar entre dos notificaciones consecutivas. Es un campo opcional, pero se ha puesto a 1 para que le dé tiempo a la aplicación de procesar las notificaciones anteriores.

Si esta suscripción perteneciese a otra familia de dispositivos, el único campo que cambiaría sería el “entities” dentro de “subject”. El campo especificaría otro prefijo distinto para seleccionar a la familia de dispositivos en cuestión.

Las otras 4 suscripciones de esta misma familia de dispositivos serían muy parecidas. Varía la descripción, los dos campos “attrs” y el campo “url”. Así se cambiaría el filtro, el cuerpo del mensaje de notificación y la URL destino, para notificar de cambios en otros atributos, enviándolos a la dirección correspondiente. En el apartado dedicado al Controlador, se profundizarán en las interfaces. Adelanto que las direcciones dedicadas a la recepción de las notificaciones son: {base}/value, {base}/ipaddress, {base}/calibration, {base}/state y {base}/location.

Dentro de la clase FiwClient hay dos métodos encargados de generar las suscripciones. En primer lugar, la función privada:

```
private Subscription subsGenerator(String description,String baseId,String url, String expires, String attr)
```

Figura 63. Función subsGenerator

Esta función genera una suscripción según el estándar NGSIv2. Es necesario especificar la descripción, el identificador de la familia de dispositivos, la URL destino, la fecha de caducidad y el atributo al que suscribirse. La implementación es trivial ya que contamos con el conjunto de clases que forman una suscripción en NGSIv2. Esta función se utiliza únicamente dentro del siguiente método público del cliente, que permite generar las 5 suscripciones por defecto:

```
public List<String> createDefaultSubscriptions(String baseId)
```

Figura 64. Función createDefaultSubscriptions

Sólo hay que especificar la familia de dispositivos sobre la que se quieren crear las suscripciones. La implementación de este método consiste en la creación de las 5 suscripciones usando el método privado mencionado arriba y su envío al Orion Context Broker. Devuelve una lista con 5 String que contiene la localización dentro del Context Broker de las suscripciones recién creadas. En caso de error en la creación, la localización será igual a null.

Para cerrar esta explicación, se muestra un mensaje de notificación como respuesta a un cambio en la posición del dispositivo en el caso del glucómetro. Como se indicó a la hora de crear la suscripción, solo se envía el atributo “location” y en formato “keyValues”.

{"subscriptionId":"5da621b46541f5201d423277","data":[{"id":"fiware_glucometer_1","type":"Device","location":{"type":"Point","coordinates":[160,110]}}]}

Figura 65. Ejemplo de notificación NGSIv2

3.6.2 Cliente

Las direcciones IP “address” y “serverAddress” son las direcciones del Orion Context Broker y el servidor donde se ejecuta la aplicación java respectivamente. Por tanto, “serverAddress” indica dónde se tienen que enviar las notificaciones. Igualmente, los puertos “port” y “serverPort” son los puertos correspondientes a las direcciones mencionadas. Estas 4 variables como se verá más adelante son modificables por el usuario a la hora de ejecutar la aplicación por la línea de comandos. Estas variables con inyectadas por tanto en tiempo de ejecución, y se le pasa a este cliente a través de su constructor. Si no se especifica ningún parámetro por la línea de comandos hay una serie de valores por defecto.

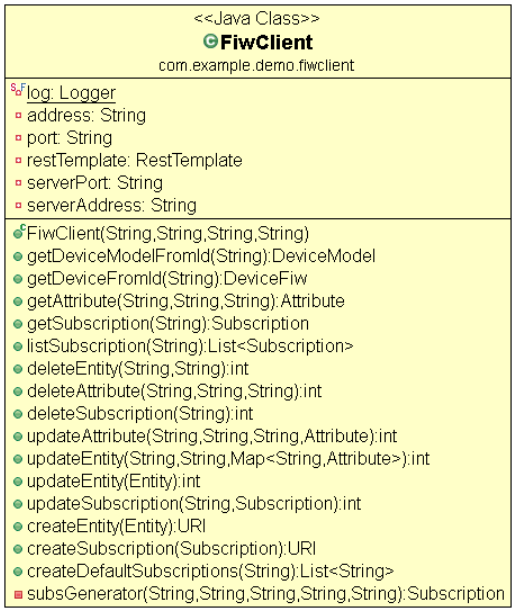

Figura 66. Diagrama de clase FiwClient

3.7 Cliente MongoDB

MongoDB es un almacén de documentos NoSQL. En este caso almacenamos dos tipos de objetos, TemplateDB y TemplateDevice. El primero almacena información general de una familia de dispositivos, mientras que el segundo sólo guarda información relevante para un dispositivo concreto.

El framework Spring usado en este proyecto, tiene una sección llamada “Spring Data MongoDB” que se centra en interaccionar con bases de datos MongoDB. Esto implica que no hay que aprender en profundidad el lenguaje usado en MongoDB, simplemente hay que escribir algunos métodos y esta librería los implementa automáticamente. Es parecido al funcionamiento de la definición del cliente FHIR explicado con anterioridad.

Hay que definir dos interfaces. Una que extienda a MongoRepository<TemplateDB, String> y otra a MongoRepository<TemplateDevice, String>. El primer valor define con qué tipo de datos va a trabajar y el segundo el tipo de dato del identificador de los recursos. Al extender MongoRepository, la interfaz viene por defecto con muchas operaciones muy útiles, incluyendo las operaciones CRUD (create-read-update-delete). Aparte de las ya incluidas, se pueden definir

métodos propios. En este caso se define uno más en cada interfaz para recuperar el recurso según su identificador.

3.7.1 TemplateDB

Esta clase especifica el contenido de la entrada de la base de datos asociada a cada familia de dispositivos. Se guardan los siguientes atributos:

- **"uniqueId":** identificador único de la familia de dispositivos.
- **"serverId":** identificador, dentro del servidor FHIR, del recurso DeviceDefinition que describe a la familia de dispositivos.
- **"deviceDefinitionTemplate":** plantilla del recurso DeviceDefinition.
- **"deviceTemplate":** plantilla del primer recurso Device (FHIR) de la familia.
- **"deviceMetricTemplate":** plantilla del recurso DeviceMetric.
- **"observationTemplate":** plantilla del recurso Observation.
- **"idList":** lista de identificadores de todos los recursos Device asociados a esta familia.
- **"subList":** lista que contiene las localizaciones de las 5 suscripciones asociadas a esta familia.

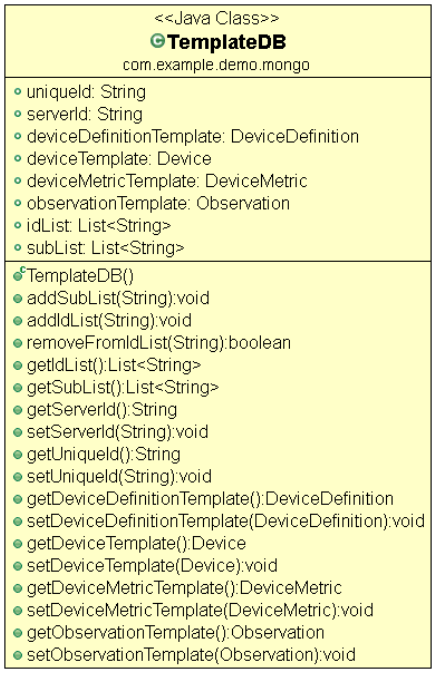

Figura 67. Diagrama de clase TemplateDB

3.7.2 TemplateDevice

Esta clase especifica el contenido de la entrada de la base de datos asociada a cada dispositivo físico. Se guardan los siguientes atributos:

- **"uniqueId":** identificador único asociado a un dispositivo físico.
- **"numberObservation":** número asociado a la última observación que realizó el dispositivo.
- **"locationServerId":** identificador, dentro del servidor FHIR, del recurso Location asociado a este dispositivo.
- **"deviceServerId":** identificador, dentro del servidor FHIR, del recurso Device que representa a este dispositivo.

- **“deviceMetricServerId”:** identificador, dentro del servidor FHIR, del recurso DeviceMetric asociado a este dispositivo.
- **“deviceTemplate”:** plantilla del recurso Device (FHIR) que representa este dispositivo.

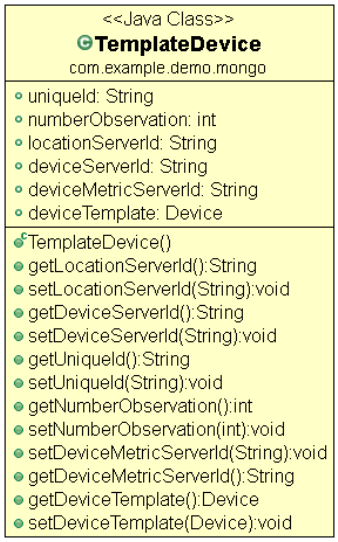

Figura 68. Diagrama de clase TemplateDevice

3.7.3 TemplateDBRepository

```
public interface TemplateDBRepository extends MongoRepository<TemplateDB, String> {

    public TemplateDB findByUniqueId(String id);

}
```
Figura 69. Interfaz TemplateDBRepository

3.7.4 TemplateDeviceRepository

```
public interface TemplateDeviceRepository extends MongoRepository<TemplateDevice, String> {

    public TemplateDevice findByUniqueId(String id);


}
```
Figura 70. TemplateDeviceRepository

3.8 RestController

Es el cerebro de la aplicación, se encarga de capturar las peticiones HTTP entrantes, de procesar los datos recibidos y de actualizar los valores de los recursos ya sea en la base de datos, en el servidor FHIR remoto o en Orion a través de los diferentes clientes de cada servicio. En definitiva, es el encargado de ofrecer la interfaz REST de la pasarela.

Cada uno de los métodos presentes en esta clase están asociados a una dirección y a un tipo de mensaje HTTP. Por ejemplo, “/value” y POST. Esto se consigue gracias a las etiquetas de Spring que se explicarán en cada método. En la siguiente tabla se muestra un resumen de la interfaz REST ofrecida:

URL	OPERACIÓN	DESCRIPCIÓN
{base}/devicedefinition	POST	Crea una nueva plantilla de tipo DeviceDefinition. En el cuerpo del mensaje se encuentra el recurso en formato JSON. El identificador se obtiene del campo identifier dentro del recurso codificado en el mensaje. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de que el recurso presente errores de sintaxis se devuelve un mensaje HTTP 400 BAD REQUEST. También se devuelve un mensaje HTTP 400 BAD REQUEST si el recurso existe ya en la pasarela.
{base}/devicedefinition/{id}	GET	Recupera la plantilla del recurso DeviceDefinition identificada con id. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de no existir ninguna plantilla que se identifique con id, se devuelve un mensaje HTTP 404 NOT FOUND.
{base}/devicedefinition/{id}	PUT	Actualiza una plantilla DeviceDefinition ya existente identificada con id. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de que el recurso presente errores de sintaxis se devuelve un mensaje HTTP 400 BAD REQUEST. También se devuelve un mensaje HTTP 400 BAD REQUEST si el recurso a actualizar no existe en la pasarela.
{base}/device	POST	Crea una plantilla de tipo Device para una nueva familia de dispositivos. En el cuerpo del mensaje se encuentra el recurso en formato JSON. El identificador se obtiene del campo identifier dentro del recurso codificado en el mensaje. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de que el recurso presente errores de sintaxis se devuelve un mensaje HTTP 400 BAD REQUEST. También se devuelve un mensaje HTTP 400 BAD REQUEST si el recurso existe ya en la pasarela.
{base}/deviceonly	POST	Crea una plantilla de tipo Device para una familia de dispositivos ya existente en el sistema. En el cuerpo del mensaje se encuentra el recurso en formato JSON. El identificador se obtiene del campo identifier dentro del recurso codificado en el mensaje. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error.

		En caso de que el recurso presente errores de sintaxis se devuelve un mensaje HTTP 400 BAD REQUEST. También se devuelve un mensaje HTTP 400 BAD REQUEST si el recurso existe ya en la pasarela.
{base}/device/{id}	GET	Recupera la plantilla del recurso Device identificada con id. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de no existir ninguna plantilla en la pasarela que se identifique con id, se devuelve un mensaje HTTP 404 NOT FOUND.
{base}/device/{id}	PUT	Actualiza una plantilla Device ya existente identificada con id. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de que el recurso presente errores de sintaxis se devuelve un mensaje HTTP 400 BAD REQUEST. También se devuelve un mensaje HTTP 400 BAD REQUEST si el recurso a actualizar no existe en la pasarela.
{base}/device/{id}	DELETE	Borra la plantilla de un recurso Device del sistema identificada con id. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de no existir ninguna entrada en el sistema identificada con id, se devuelve un mensaje HTTP 400 BAD REQUEST.
{base}/devicemetric	POST	Crea una nueva plantilla de tipo DeviceMetric. En el cuerpo del mensaje se encuentra el recurso en formato JSON. El identificador se obtiene del campo identifier dentro del recurso codificado en el mensaje. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de que el recurso presente errores de sintaxis se devuelve un mensaje HTTP 400 BAD REQUEST. También se devuelve un mensaje HTTP 400 BAD REQUEST si el recurso existe ya en la pasarela.
{base}/devicemetric/{id}	GET	Recupera la plantilla del recurso DeviceMetric identificada con id. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de no existir ninguna plantilla que se identifique con id, se devuelve un mensaje HTTP 404 NOT FOUND.
{base}/devicemetric/{id}	PUT	Actualiza una plantilla DeviceMetric ya existente identificada con id. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de que el recurso presente errores de sintaxis se devuelve un mensaje HTTP 400

		BAD REQUEST. También se devuelve un mensaje HTTP 400 BAD REQUEST si el recurso a actualizar no existe en el sistema.
{base}/observation	POST	Crea una nueva plantilla de tipo Observation. En el cuerpo del mensaje se encuentra el recurso en formato JSON. El identificador se obtiene del campo identifier dentro del recurso codificado en el mensaje. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de que el recurso presente errores de sintaxis se devuelve un mensaje HTTP 400 BAD REQUEST. También se devuelve un mensaje HTTP 400 BAD REQUEST si el recurso existe ya en la pasarela.
{base}/observation/{id}	GET	Recupera la plantilla del recurso Observation identificada con id. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de no existir ninguna plantilla que se identifique con id, se devuelve un mensaje HTTP 404 NOT FOUND.
{base}/observation/{id}	PUT	Actualiza una plantilla Observation ya existente identificada con id. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de que el recurso presente errores de sintaxis se devuelve un mensaje HTTP 400 BAD REQUEST. También se devuelve un mensaje HTTP 400 BAD REQUEST si el recurso a actualizar no existe en la pasarela.
{base}/devicefamily/{id}	DELETE	Borra todas las plantillas pertenecientes a una familia de dispositivos identificada con id. Devuelve un mensaje HTTP 200 OK si no ocurre ningún error. En caso de no existir ninguna entrada en el sistema identificada con id, se devuelve un mensaje HTTP 400 BAD REQUEST. Recordar que una familia de dispositivos consiste como mínimo de los recursos DeviceDefinition, DeviceMetric, Observation y Device.

Tabla 132. Resumen interfaz REST

Esta tabla muestra la interfaz encargada de capturar las notificaciones procedentes de Orion.

URL	OPERACIÓN	DESCRIPCIÓN
{base}/value	POST	Procesa la notificación asíncrona de Orion como consecuencia del cambio en el valor medido de un dispositivo.
{base}/state	POST	Procesa la notificación asíncrona de Orion como consecuencia del cambio en el estado de un dispositivo.
{base}/location	POST	Procesa la notificación asíncrona de Orion como consecuencia del cambio en la posición de un dispositivo.
{base}/calibration	POST	Procesa la notificación asíncrona de Orion como consecuencia del cambio en la calibración de un dispositivo.
{base}/ipaddress	POST	Procesa la notificación asíncrona de Orion como consecuencia del cambio en la dirección IP de un dispositivo.

Tabla 133. Interfaz para capturar notificaciones

3.8.1 Nueva familia de dispositivos fase 1: Creación de plantilla DeviceDefinition

Consiste en el paso inicial para crear las plantillas de una nueva familia de dispositivos en la pasarela. Se crea un nuevo recurso DeviceDefinition. Se envía por la línea de comandos, o por una aplicación como Postman, un mensaje POST con el recurso ya sea en formato texto plano o JSON. La URL destino es de la forma, "{base}/devicedefinition".

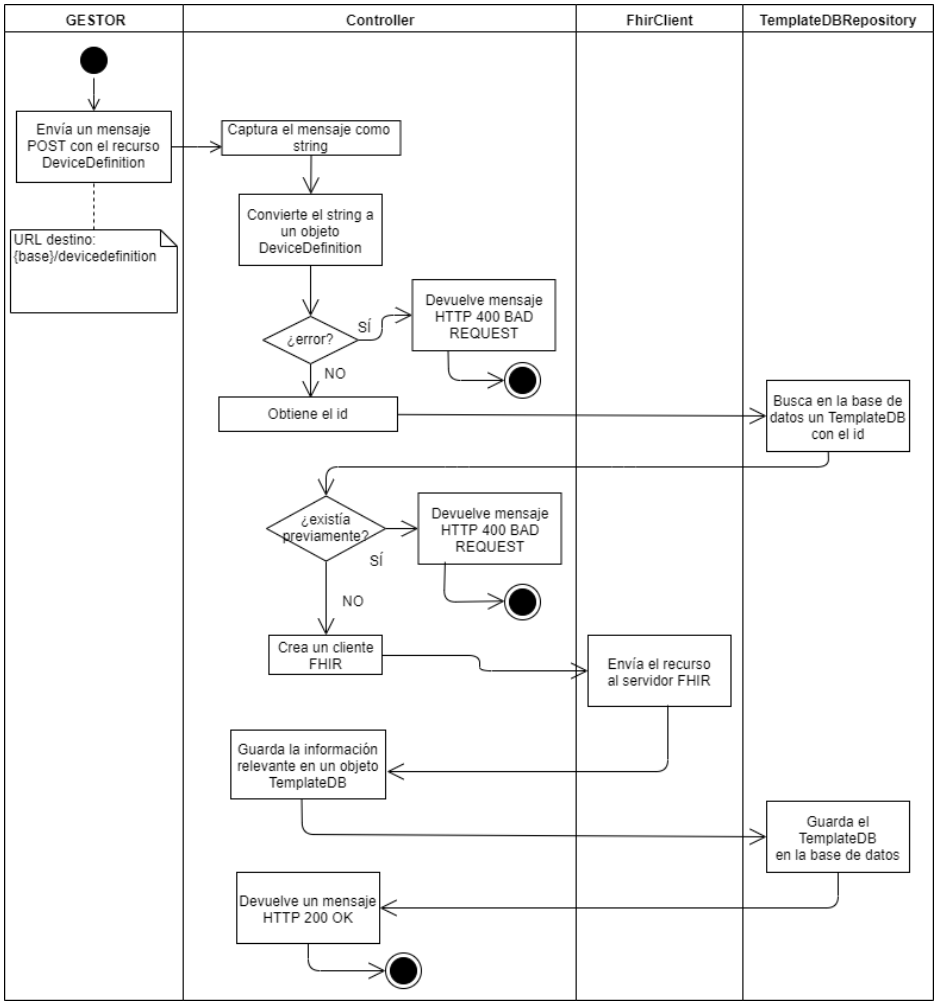

Figura 71. Diagrama de actividad creación plantilla DeviceDefinition

En la figura de abajo, se observa como mediante la etiqueta @PostMapping se indica que se espera un mensaje POST, le indicamos también la url (“/devicedefinition”) y que el cuerpo del mensaje puede estar codificado en JSON o en texto plano. Además, usando la etiqueta @RequestBody indicamos que el recurso codificado en el cuerpo del mensaje recibido tiene que ser convertido en un String.

```
@PostMapping(value = "/devicedefinition",
        consumes = {
                MediaType.TEXT_PLAIN_VALUE,
                MediaType.APPLICATION_JSON_VALUE
        })
public ResponseEntity<HttpStatus> deviceDefinitionCreation(@RequestBody String template){
```

Figura 72. Etiquetas Spring creación DeviceDefinition

Lo primero que hay que hacer es convertir el texto plano o el JSON recibido en la petición a un objeto de tipo DeviceDefinition. Hay que usar un objeto del tipo IParser de la librería HAPI FHIR, se obtiene del objeto de contexto FHIR.

Se utiliza un bloque try/catch para capturar los posibles errores del recurso enviado. En caso de tener errores de sintaxis, se enviaría un mensaje HTTP BAD REQUEST al usuario con un mensaje indicando que el recurso enviado no es correcto.

A continuación, hay que comprobar que el recurso enviado no exista ya en el sistema. En caso de haber sido creado anteriormente, se envía al usuario un mensaje HTTP 400 BAD REQUEST con un mensaje indicando el error.

En el caso de que no exista en el sistema, se procede a crear un cliente de FHIR y enviar el recurso al servidor. Además, se crea una nueva plantilla y se guarda dentro el id del recurso en el servidor, el id del recurso en mi sistema y la plantilla del DeviceDefinition recibida. Finalmente, se introduce esta nueva plantilla en la base de datos y se contesta al usuario con un mensaje de éxito.

3.8.2 Nueva familia de dispositivos fase 2: Creación de plantilla Device

Consiste en el segundo paso para crear una nueva familia de dispositivos en la pasarela. Se envía por la línea de comandos, o por una aplicación como Postman, un mensaje POST con el recurso Device ya sea en formato texto plano o JSON. La URL destino es de la forma, "{base}/device".

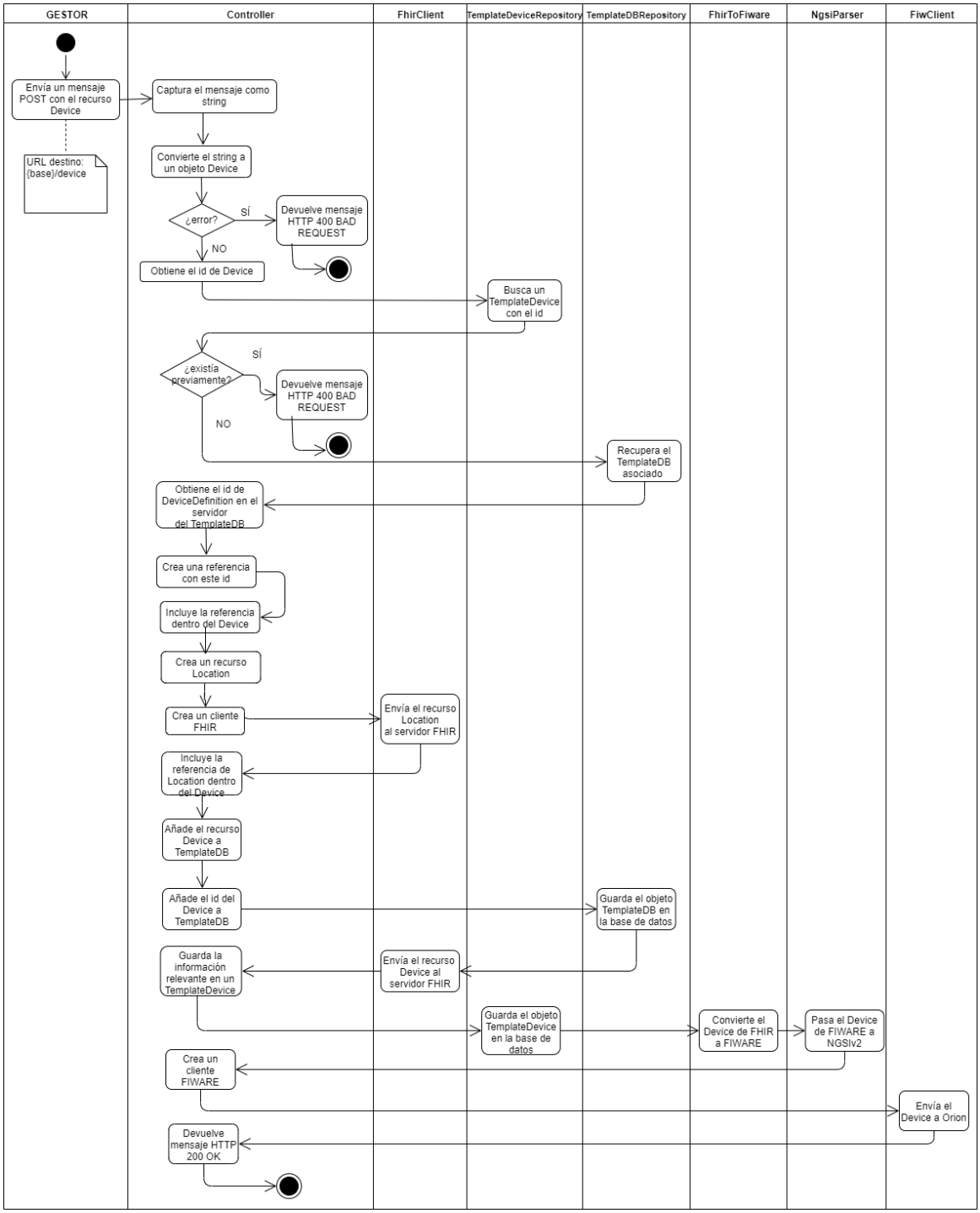

Figura 73. Diagrama de actividad creación plantilla Device

En la figura de abajo, se observa que mediante la etiqueta @PostMapping se indica que se espera un mensaje POST, se indica también la url ("/device") y que el cuerpo del mensaje puede estar codificado en JSON o como texto plano. Además, usando la etiqueta @RequestBody se indica que el recurso codificado en el cuerpo del mensaje recibido tiene que ser convertido en un String.

```
@PostMapping(value = "/device",
        consumes = {
                MediaType.APPLICATION_JSON_VALUE,
                MediaType.TEXT_PLAIN_VALUE
        })
public ResponseEntity<String> deviceCreation(@RequestBody String template){
```

Figura 74. Etiquetas Spring creación Device

Lo primero que se hace es convertir el texto plano o el JSON a un objeto de tipo Device. Debemos usar un objeto del tipo IParser de la librería HAPI FHIR, se obtiene del objeto de contexto FHIR.

Se utiliza un bloque try/catch para capturar los posibles errores del recurso enviado. En caso de tener errores de sintaxis, se envía un mensaje HTTP 400 BAD REQUEST al usuario con un mensaje indicando que el recurso enviado no es correcto.

A continuación, hay que comprobar que el recurso enviado no exista ya en el sistema. En caso de haber sido creado anteriormente, se envía al usuario un mensaje HTTP 400 BAD REQUEST con un mensaje indicando el error.

Ahora, se recupera la dirección del DeviceDefinition asociado a este Device de la base de datos y se introduce como una referencia dentro del objeto Device.

Hay que crear un recurso Location y un cliente FHIR para subirlo posteriormente al servidor. El identificador de este recurso en el servidor hay que guardarlo como una referencia dentro del recurso Device en el campo location.

Hay que guardar la plantilla del Device en la base de datos, así como añadir el identificador de este recurso Device a la lista de identificadores de esta familia de dispositivos. Esto se hace por si quisiera borrar todas las entradas de la base de datos asociadas a una familia de dispositivos.

Posteriormente se sube el recurso Device al servidor FHIR y se crea una nueva entrada en la base de datos exclusivamente para este dispositivo en concreto. Aquí se guarda el identificador único del dispositivo, los identificadores de los recursos Device y Location en el servidor FHIR y se establece a cero el número de observaciones asociadas a este dispositivo.

A continuación, se convierte este recurso Device a un recurso Device del mundo FIWARE. Tras esto, se pasa al estándar NGSIv2 para poder enviarlo al Context Broker. Se crea un cliente FIWARE y se envía el recurso en formato NGSIv2 al Broker. Tras esto, se devuelve un mensaje de éxito al usuario.

3.8.3 Nueva familia de dispositivos fase 3: Creación de plantilla DeviceMetric

Crea un nuevo recurso DeviceMetric, consiste en el tercer paso para crear una nueva familia de dispositivos. Se envía por la línea de comandos, o por una aplicación como Postman, un mensaje POST con el recurso ya sea en formato texto plano o JSON. La URL destino es de la forma "{base}/devicemetric".

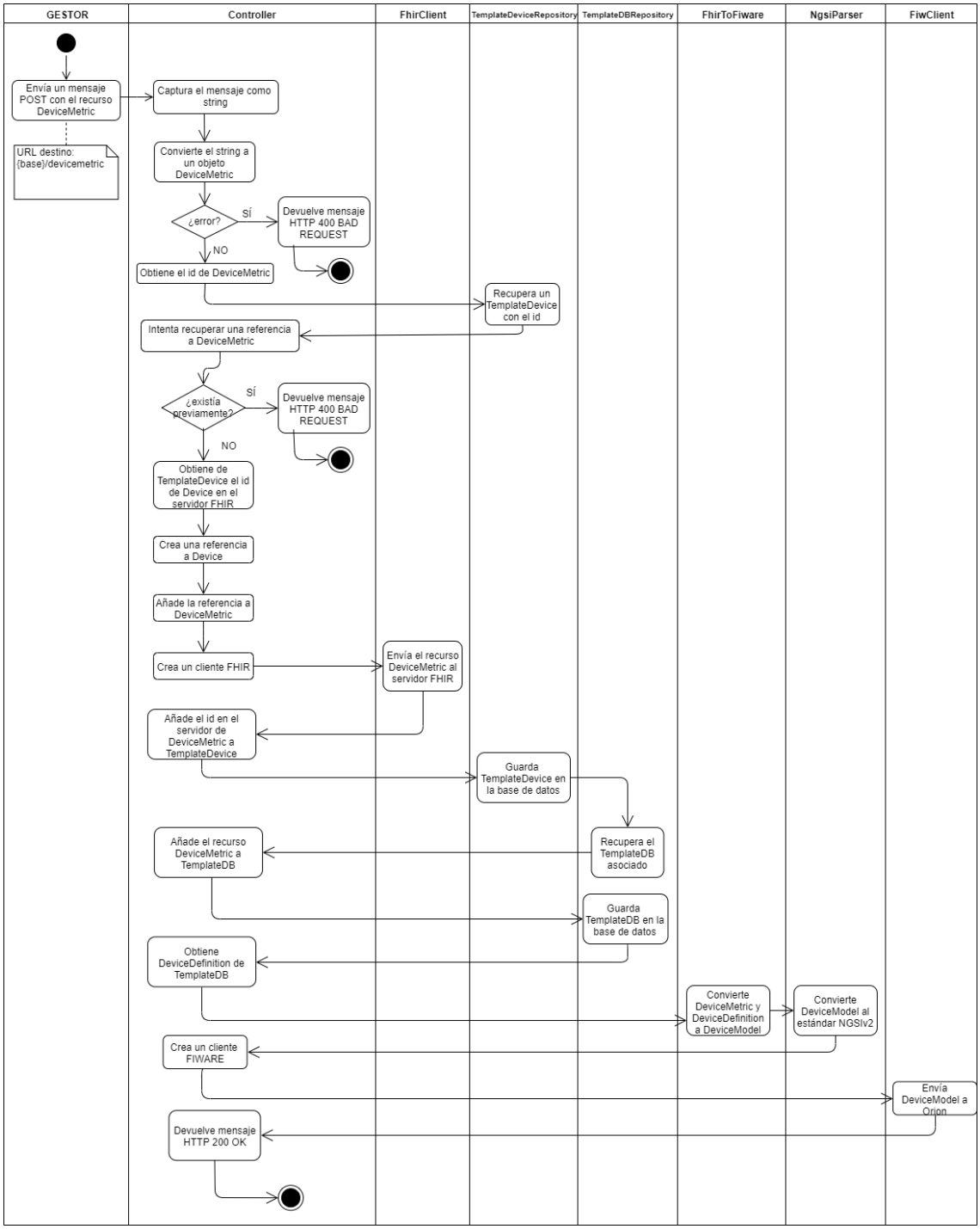

Figura 75. Diagrama de actividad creación plantilla DeviceMetric

Mediante la etiqueta @PostMapping se indica que esperamos recibir un mensaje POST, se indica también la URL ("/devicemetric") y que el cuerpo del mensaje puede estar codificado en JSON o como texto plano. Además, usando la etiqueta @RequestBody se indica que el recurso codificado en el cuerpo del mensaje recibido tiene que ser convertido en un String.

```
@PostMapping(value = "/devicemetric",
        consumes = {
                MediaType.APPLICATION_JSON_VALUE,
                MediaType.TEXT_PLAIN_VALUE
        })
public ResponseEntity<String> deviceMetricCreation(@RequestBody String template){
```

Figura 76. Etiquetas Spring creación DeviceMetric

Lo primero que se hace es convertir el texto plano o el JSON a un objeto de tipo DeviceMetric. Hay que usar un objeto del tipo IParser de la librería HAPI FHIR, se obtiene del objeto de contexto FHIR.

Se utiliza un bloque try/catch para capturar los posibles errores del recurso enviado. En caso de tener errores de sintaxis, se envía un mensaje HTTP BAD REQUEST al usuario con un mensaje indicando que el recurso enviado no es correcto.

A continuación, hay que comprobar que el recurso enviado no exista ya en el sistema. En caso de haber sido creado anteriormente, se envía al usuario un mensaje HTTP BAD REQUEST con un mensaje indicando el error.

Ahora, se obtiene de la base de datos el identificador del recurso Device dentro del servidor FHIR. Con este identificador se crea una referencia al recurso que se mete dentro del DeviceMetric.
Se crea un cliente FHIR y se sube el DeviceMetric al servidor FHIR. Como consecuencia se crea un identificador del DeviceMetric en el servidor que hay que guardar en la base de datos para su posterior uso.

Posteriormente, se guarda la plantilla del DeviceMetric en la base de datos asociada a la familia de dispositivos. Para terminar, hay que crear y enviar el recurso DeviceModel de FIWARE. Se genera a partir del recurso DeviceDefinition y DeviceMetric de FHIR. Por lo que hay que recuperar el DeviceDefinition de la base de datos. Así se genera la entidad DeviceModel, que tras pasar al estándar NGSIv2, se envía al Context Broker de FIWARE. Tras esto, se envía un mensaje de éxito al usuario.

3.8.4 Nueva familia de dispositivos fase 4: Creación de plantilla Observation

Este método se usa para crear un nuevo recurso Observation, es el paso final en el proceso de creación de una nueva familia. Se envía por la línea de comandos, o por una aplicación como Postman, un mensaje POST con el recurso ya sea en formato texto plano o JSON. La URL destino es de la forma “{base}/observation”.

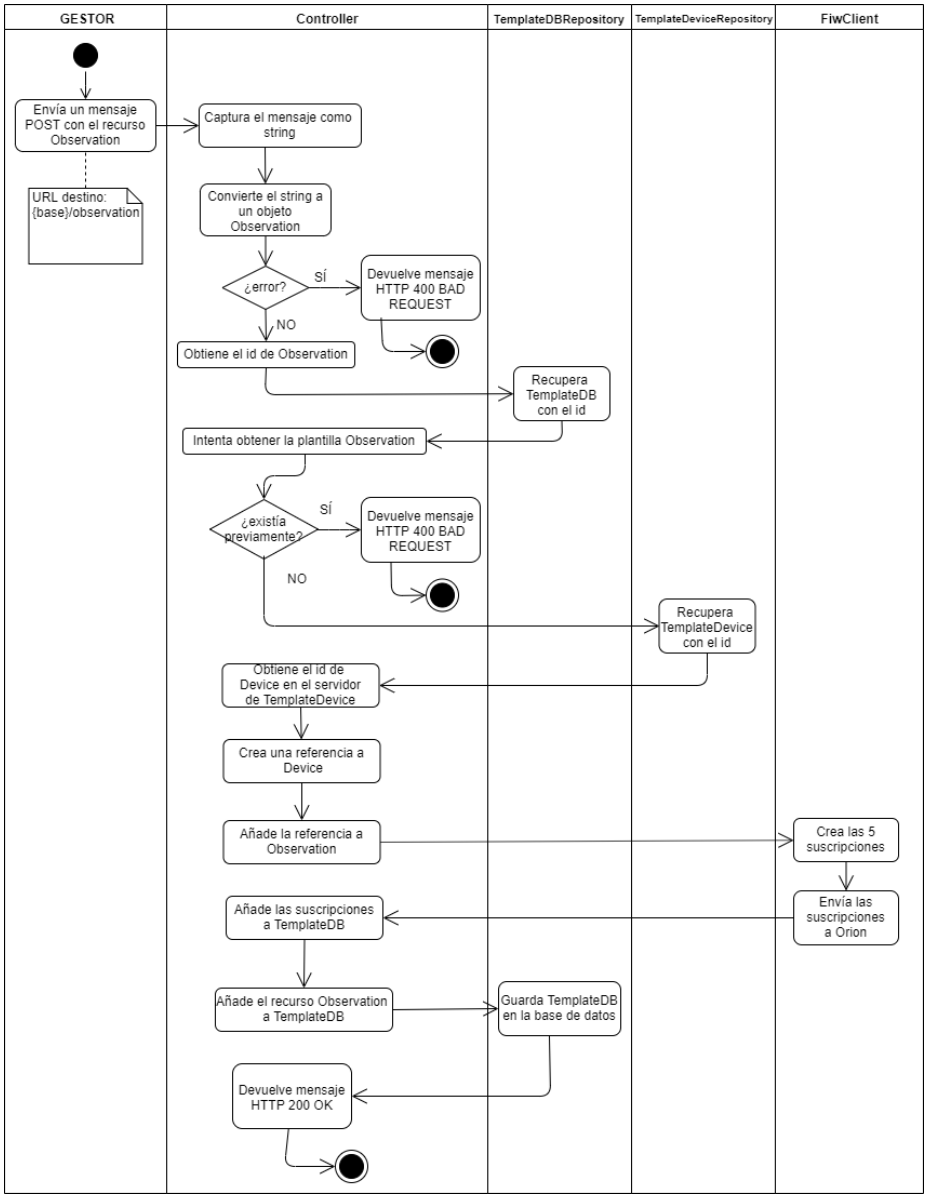

Figura 77. Diagrama de actividad creación plantilla Observation

Mediante la etiqueta @PostMapping se indica que se espera un mensaje POST, se indica también la url ("/observation") y que el cuerpo del mensaje puede estar codificado en JSON o como texto plano. Además, usando la etiqueta @RequestBody se indica que el recurso codificado en el cuerpo del mensaje recibido tiene que ser convertido en un String.

```
@PostMapping(value = "/observation",
        consumes = {
                MediaType.APPLICATION_JSON_VALUE
        })
public ResponseEntity<String> observationCreation(@RequestBody String template){
```

Figura 78. Etiquetas Spring creación Observation

Lo primero que se hace es convertir el texto plano o el JSON a un objeto de tipo Observation. Hay que usar un objeto del tipo IParser de la librería HAPI FHIR, se obtiene del objeto de contexto FHIR.

Se utiliza un bloque try/catch para capturar los posibles errores del recurso enviado. En caso de tener errores de sintaxis, se enviaría un mensaje HTTP BAD REQUEST al usuario con un mensaje indicando que el recurso enviado no es correcto.

A continuación, hay que comprobar que el recurso enviado no exista ya en el sistema. En caso de haber sido creado anteriormente, se envía al usuario un mensaje HTTP BAD REQUEST con un mensaje indicando el error.

Ahora, se recupera de la base de datos el identificador del recurso Device en el servidor FHIR. Se crea una nueva referencia y se introduce en el campo “subject” del recurso Observation.

Para terminar, se crean las 5 suscripciones pertinentes en FIWARE asociadas a esta familia concreta de dispositivos. Inicialmente, solo se pensaba usar suscripciones genéricas. Pero en la práctica daban error ya que, al crear nuevas familias de dispositivos, saltaban las suscripciones antes de tiempo y generaban errores en el sistema. No existe una opción en FIWARE para evitar la notificación en el caso de creación de nuevas entidades, por lo que crear un conjunto de suscripciones para cada familia de dispositivos era la mejor opción. Tras crear las suscripciones se envía un mensaje de éxito al usuario. Además, se van guardando en una entrada de la base de datos porque harán falta en el caso de que el usuario quiera eliminar la familia de dispositivos del sistema.

Finalmente, se guarda en la base de datos la entrada con las suscripciones ante mencionadas y el recurso Observation, se envía al usuario un mensaje HTTP 200 OK y un breve mensaje de éxito.

3.8.5 Creación de plantilla Device de una familia ya existente.

Se usa para crear un nuevo recurso Device perteneciente a alguna familia de dispositivos ya asentada en el sistema. La diferencia radica en que ya se disponen de las plantillas para los recursos DeviceMetric, DeviceDefiniton y Observation asociadas a esta familia por lo que solo se necesita un nuevo Device para crear toda la estructura tanto en FHIR como FIWARE.

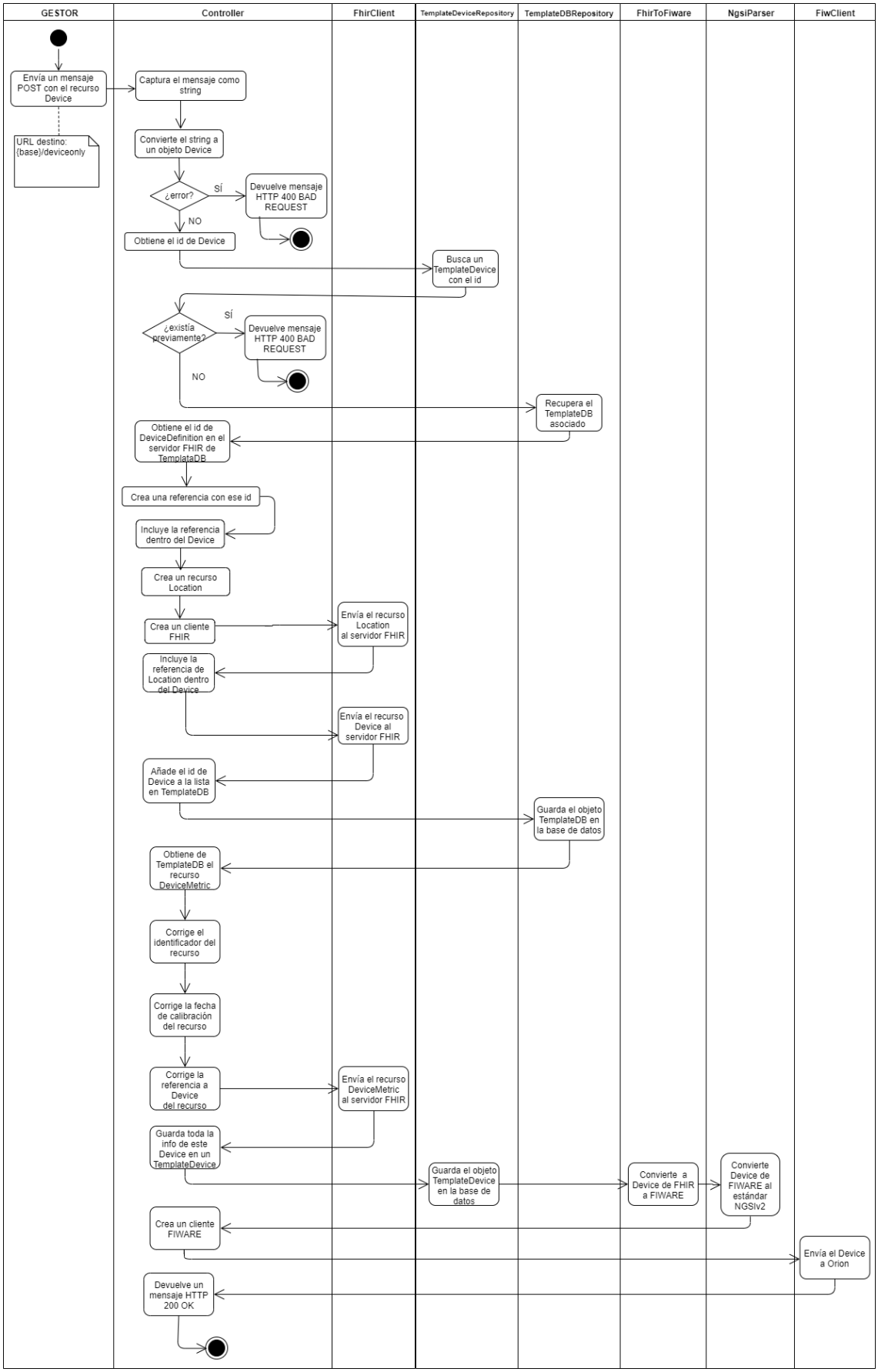

Figura 79. Diagrama de actividad creación plantilla Device adicional

Se envía por la línea de comandos, o por una aplicación como Postman, un mensaje POST con el recurso ya sea en formato texto plano o JSON. La URL destino es de la forma "{base}/deviceonly".

Mediante la etiqueta @PostMapping se indica que se espera recibir un mensaje POST, se indica también la url ("/deviceonly") y que el cuerpo del mensaje puede estar codificado en JSON o como texto plano. Además, usando la etiqueta @RequestBody se indica que el recurso codificado en el cuerpo del mensaje recibido tiene que ser convertido en un String.

```
@PostMapping(value = "/deviceonly",
        consumes = {
                MediaType.APPLICATION_JSON_VALUE,
                MediaType.TEXT_PLAIN_VALUE
        })
public ResponseEntity<String> deviceCreationQuick(@RequestBody String template){
```

Figura 80. Etiquetas Spring creación Device adicional

Lo primero que se hace es convertir el texto plano o el JSON a un objeto de tipo Device. Hay que usar un objeto del tipo IParser de la librería HAPI FHIR, se obtiene del objeto de contexto FHIR.

Se utiliza un bloque try/catch para capturar los posibles errores del recurso enviado. En caso de tener errores de sintaxis, se enviaría un mensaje HTTP 400 BAD REQUEST al usuario indicando que el recurso enviado no es correcto.

A continuación, hay que comprobar que el recurso enviado no exista ya en el sistema. En caso de haber sido creado anteriormente, se envía al usuario un mensaje HTTP 400 BAD REQUEST indicando el error.

Ahora, se recupera la dirección del DeviceDefinition asociado a este Device de la base de datos y se introduce como una referencia dentro del objeto Device.

Hay que crear un recurso Location y crear un cliente FHIR para subirlo posteriormente al servidor. El identificador de este recurso en el servidor hay que guardarlo como una referencia dentro del recurso Device en el campo location.

Posteriormente se sube el recurso Device al servidor FHIR. Además, se añade el identificador de este recurso a la lista de identificadores asociados a esta familia de dispositivos para su posterior uso.

A continuación, se monta el recurso DeviceMetric a partir de la plantilla y se sube al servidor. A la plantilla se le añade el identificador correspondiente, la nueva fecha de calibración, y la referencia correcta al recurso Device. No hay que hacer nada con la plantilla Observation, ya que la primera observación se generará correctamente cuando se actualice el sensor en el mundo FIWARE. Finalmente, se sube el recurso al servidor FHIR.

Ahora, se crea una nueva entrada en la base de datos exclusivamente para este dispositivo en concreto. Aquí se guarda el identificador único del dispositivo, los identificadores de los recursos Device y Location en el servidor FHIR y se pone el número de observaciones asociadas a este dispositivo a uno. También se guardan el identificador del DeviceMetric en el servidor FHIR y la plantilla de este recurso para su posterior uso.

Se convierte este recurso Device a un recurso Device del mundo FIWARE. Tras esto, se convierte al estándar NGSIv2 para poder enviarlo al Context Broker. Se crea un cliente FIWARE y se envía el recurso en formato NGSIv2 al Broker. Finalmente, se devuelve un mensaje de éxito al usuario.

3.8.6 Recuperación de plantilla del recurso DeviceDefinition por su identificador

Este método se usa para recuperar la plantilla de un recurso DeviceDefinition previamente creado en el sistema usando su identificador. El gestor envía por la línea de comandos, o por una aplicación como Postman, un mensaje GET indicando el identificador de la familia del recurso a recuperar en la propia URL de la forma, "{base}/devicedefinition/{id}".

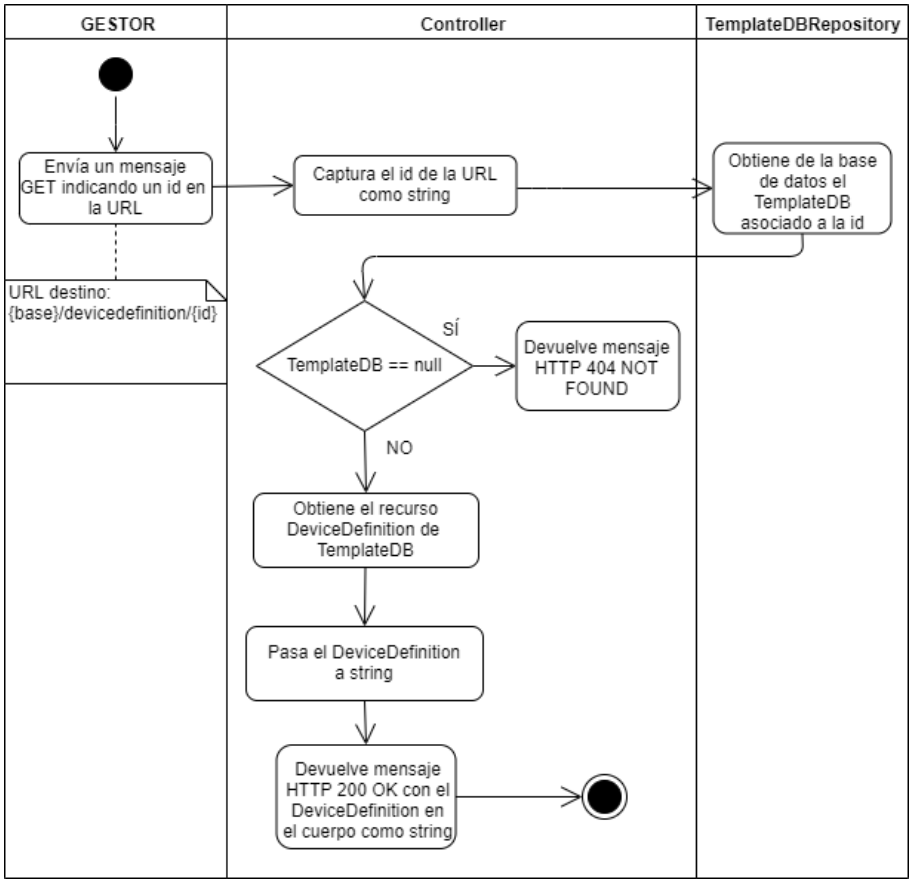

Figura 81. Diagrama de actividad recuperación plantilla DeviceDefinition

Mediante la etiqueta @GetMapping se indica que se espera recibir un mensaje GET, se indica también la URL que es de la forma ("/devicedefinition/{id}"). Con la etiqueta @PathVariable se obtiene el identificador de la URL y se convierte a un String. En caso de no existir ningún recurso con el identificador seleccionado se responde al usuario con un mensaje HTTP 404 NOT FOUND. Si existiera se devuelve al usuario un mensaje HTTP 200 OK con el recurso DeviceDefinition seleccionado en el cuerpo del mismo en formato JSON.

```
//DeviceDefinition
 @GetMapping(value = "/devicedefinition/{id}")
 public ResponseEntity<String> getDeviceDefinition(@PathVariable("id") String id){
```

Figura 82. Etiquetas Spring recuperación DeviceDefinition

3.8.7 Recuperación de plantilla del recurso Device por su identificador

Este método se usa para recuperar la plantilla de un recurso Device previamente creado en el sistema usando su identificador. El gestor envía por la línea de comandos, o por una aplicación como Postman, un mensaje GET indicando el identificador del recurso Device a recuperar en la propia URL. La URL es de la forma "{base}/device/{id}".

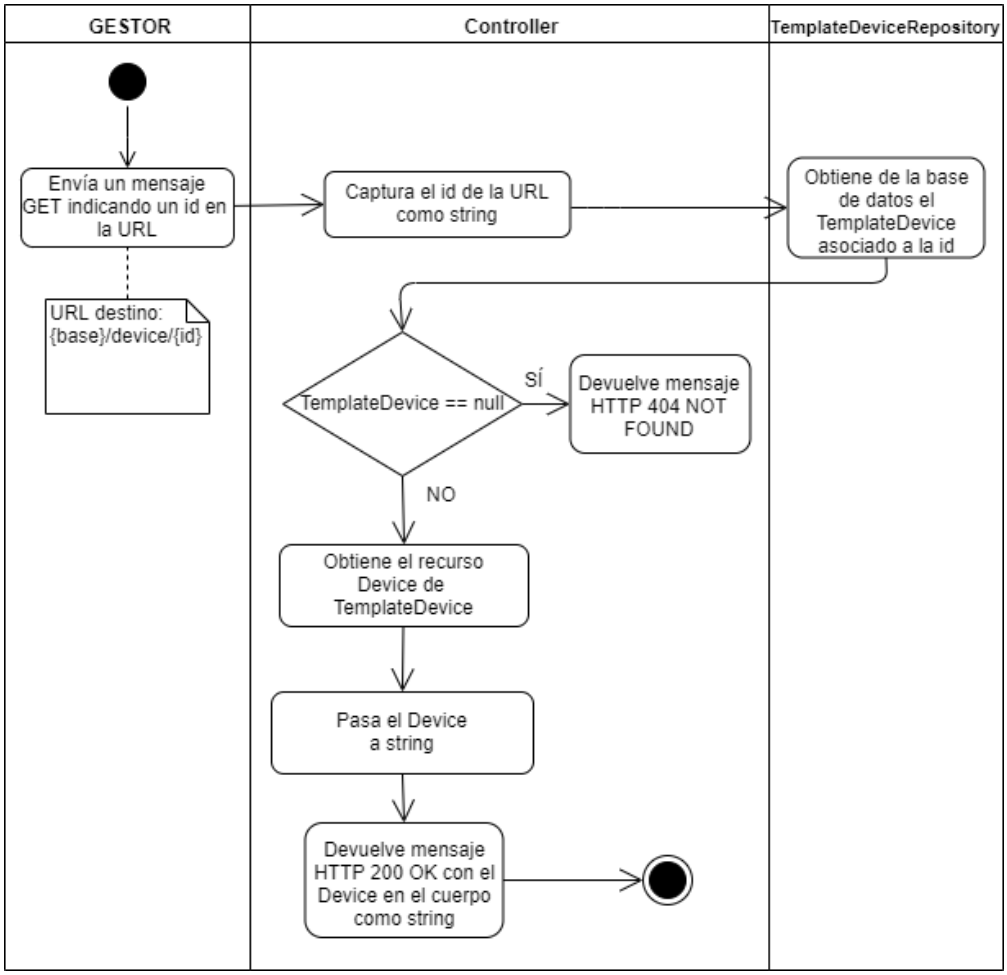

Figura 83. Diagrama de actividad recuperación plantilla Device

Mediante la etiqueta @GetMapping se indica que vamos a recibir un mensaje GET, se indica también la URL que es de la forma ("/device/{id}"). Con la etiqueta @PathVariable se obtiene el identificador de la URL y se convierte a un String. En caso de no existir ningún recurso con el identificador seleccionado se responde al usuario con un mensaje HTTP 404 NOT FOUND. Si existiera se devuelve al usuario un mensaje HTTP 200 OK con el recurso Device seleccionado en el cuerpo del mismo en formato JSON.

```
//Device
 @GetMapping(value = "/device/{id}")
 public ResponseEntity<String> getDevice(@PathVariable("id") String id)
```

Figura 84. Etiquetas Spring recuperación Device

3.8.8 Recuperación de plantilla del recurso DeviceMetric por su identificador

Este método se usa para recuperar la plantilla de un recurso DeviceMetric previamente creado en el sistema usando su identificador. El gestor envía por la línea de comandos, o por una aplicación como Postman, un mensaje GET indicando el identificador de la familia del recurso a recuperar en la propia URL. La URL es de la forma "{base}/devicemetric/{id}".

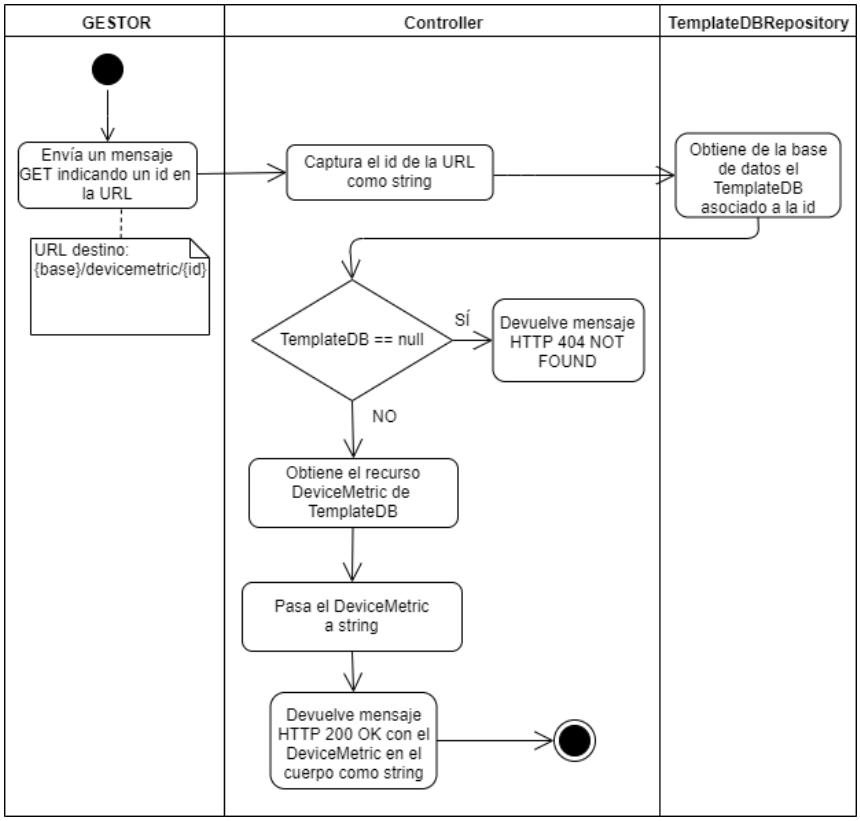

Figura 85. Diagrama de actividad recuperación plantilla DeviceMetric

Este método se usa para recuperar la plantilla de un recurso DeviceMetric previamente creado en el sistema usando su identificador. El gestor envía por la línea de comandos, o por una aplicación como Postman, un mensaje GET indicando el identificador de la familia del recurso a recuperar en la propia URL. La URL es de la forma "{base}/devicemetric/{id}"

Mediante la etiqueta @GetMapping indicamos que se espera recibir un mensaje GET, se indica también la URL que es de la forma ("/devicemetric/{id}"). Con la etiqueta @PathVariable se obtiene el identificador de la URL y se convierte a un String. En caso de no existir ningún recurso con el identificador seleccionado se responde al usuario con un mensaje HTTP 404 NOT FOUND. Si existiera se devuelve al usuario un mensaje HTTP 200 OK con el recurso DeviceMetric seleccionado en el cuerpo del mismo en formato JSON.

```
//DeviceMetric
 @GetMapping(value = "/devicemetric/{id}")
 public ResponseEntity<String> getDeviceMetric(@PathVariable("id") String id){
```

Figura 86. Etiquetas Spring recuperación DeviceMetric

3.8.9 Recuperación de plantilla del recurso Observation por su identificador

Este método se usa para recuperar la plantilla de un recurso Observation previamente creado en el sistema usando su identificador. El gestor envía por la línea de comandos, o por una aplicación como Postman, un mensaje GET indicando el identificador de la familia del recurso a recuperar en la propia URL. La URL es de la forma "{base}/observation/{id}".

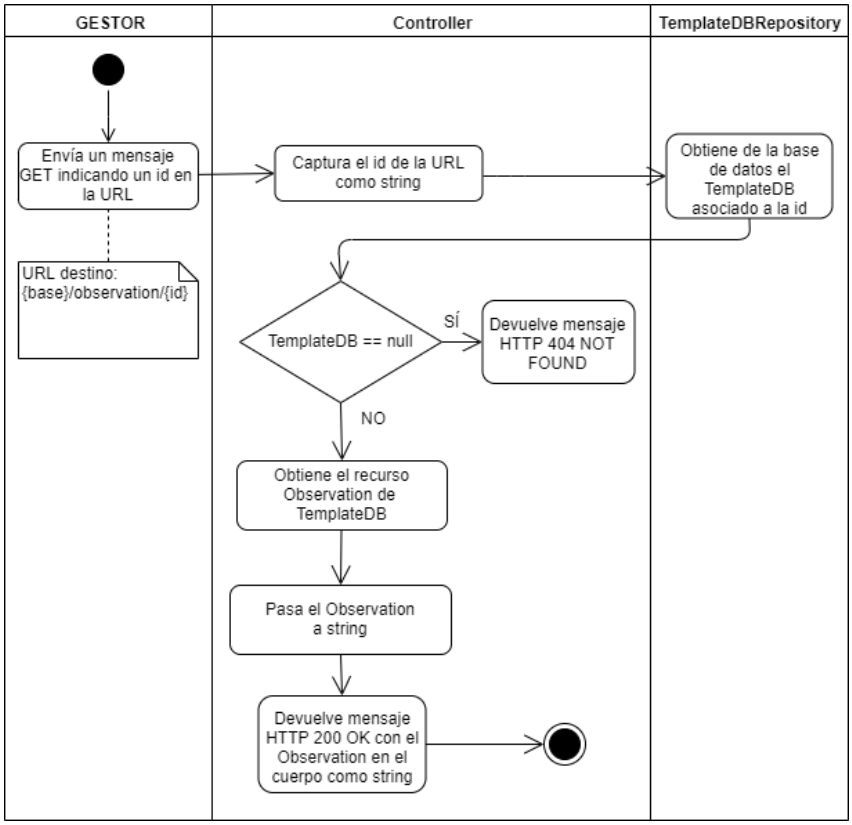

Figura 87. Diagrama de actividad recuperación plantilla Observation

Mediante la etiqueta @GetMapping se indica que se espera recibir un mensaje GET, se indica también la URL que es de la forma ("/observation/{id}"). Con la etiqueta @PathVariable se obtiene el identificador de la URL y se convierte a un String. En caso de no existir ningún recurso con el identificador seleccionado se responde al usuario con un mensaje HTTP 404 NOT FOUND. Si existiera se devuelve al usuario un mensaje HTTP 200 OK con el recurso Observation seleccionado en el cuerpo del mismo en formato JSON.

```
//Observation
 @GetMapping(value = "/observation/{id}")
 public ResponseEntity<String> getObservation(@PathVariable("id") String id){
```

Figura 88. Etiquetas Spring recuperación Observation

3.8.10 Actualización de plantilla del recurso DeviceDefinition

Este método se usa para actualizar una plantilla de recurso DeviceDefinition previamente creada en nuestro sistema. El gestor envía por la línea de comandos, o por una aplicación como Postman, un mensaje PUT indicando el identificador de la familia del recurso a actualizar en la propia URL. La URL es de la forma "{base}/devicedefinition/{id}". El nuevo recurso DeviceDefinition se envía en el cuerpo del mensaje, ya sea en formato texto plano o JSON.

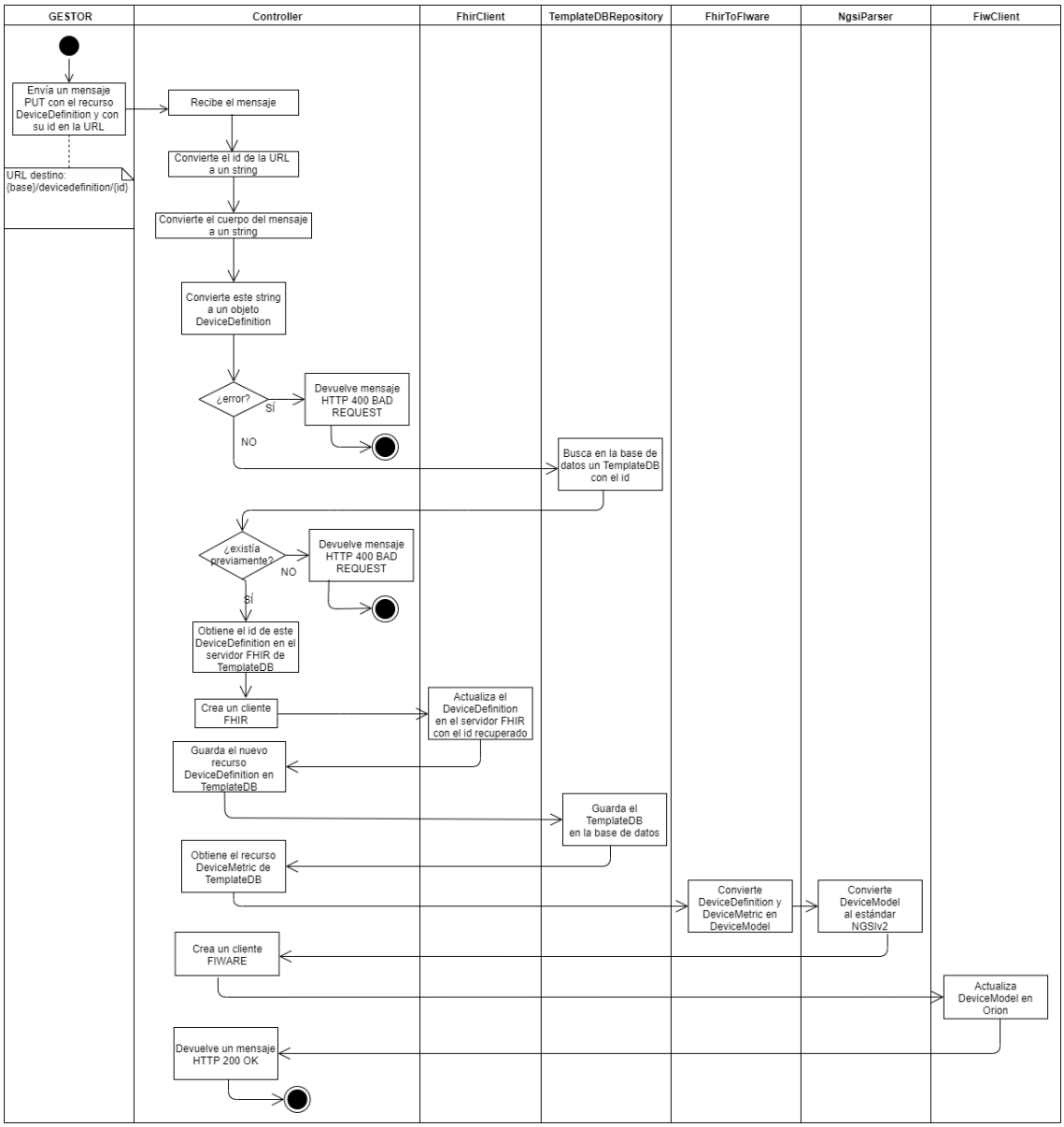

Figura 89. Diagrama de actividad actualización plantilla DeviceDefinition

Mediante la etiqueta @PutMapping se indica que se espera recibir un mensaje PUT, se indica también la URL ("/devicedefinition/{id}") y que el cuerpo del mensaje puede estar codificado en JSON o como texto plano. Con la etiqueta @PathVariable se obtiene el identificador de la URL y se convierte a un String. Además, usando la etiqueta @RequestBody se indica que el recurso DeviceDefinition codificado en el cuerpo del mensaje recibido tiene que ser convertido en un String.

```
@PutMapping(value = "/devicedefinition/{id}",
            consumes = {
                    MediaType.TEXT_PLAIN_VALUE,
                    MediaType.APPLICATION_JSON_VALUE
            })
public ResponseEntity<String> deviceDefinitionUpdate(@RequestBody String template, @PathVariable("id") String id){
```

Figura 90. Etiquetas Spring actualización DeviceDefinition

Lo primero que se hace es convertir el texto plano o el JSON a un objeto de tipo DeviceDefinition. Hay que usar un objeto del tipo IParser de la librería HAPI FHIR, se obtiene del objeto de contexto FHIR.

Se utiliza un bloque try/catch para capturar los posibles errores del recurso enviado. En caso de tener errores de sintaxis, se enviaría un mensaje HTTP 400 BAD REQUEST al usuario con un mensaje indicando que el recurso enviado no es correcto.

A continuación, hay que comprobar que el recurso enviado exista en el sistema. En caso de no haber sido creado anteriormente, se envía al usuario un mensaje HTTP 400 BAD REQUEST con un mensaje indicando el error.

En el caso de que exista en el sistema, se obtiene de la base de datos el identificador del recurso en el servidor, se crea un cliente de FHIR y se envía el recurso al servidor. Además, se actualiza la plantilla que tenía almacenada en mi base de datos con los nuevos datos.

Ahora hay que crear un recurso equivalente en el mundo FIWARE llamado DeviceModel. Para ello se recupera primero el recurso DeviceMetric de la base de datos. Con este recurso recién recuperado y el recurso DeviceDefinition que mandó el usuario se genera el recurso DeviceModel. Para terminar, se pasa este recurso al estándar NGSIv2, se crea un cliente FIWARE y se actualiza el recurso en el Context Broker. Si todo va bien, se responde al usuario con un mensaje HTTP 200 OK y un breve mensaje de éxito.

3.8.11 Actualización de plantilla del recurso Device

Este método se usa para actualizar una plantilla de recurso Device previamente creada en nuestro sistema. El gestor envía por la línea de comandos, o por una aplicación como Postman, un mensaje PUT indicando el identificador del recurso a actualizar en la propia URL. La URL es de la forma "{base}/device/{id}". El nuevo recurso Device se envía en el cuerpo del mensaje, ya sea en formato texto plano o JSON.

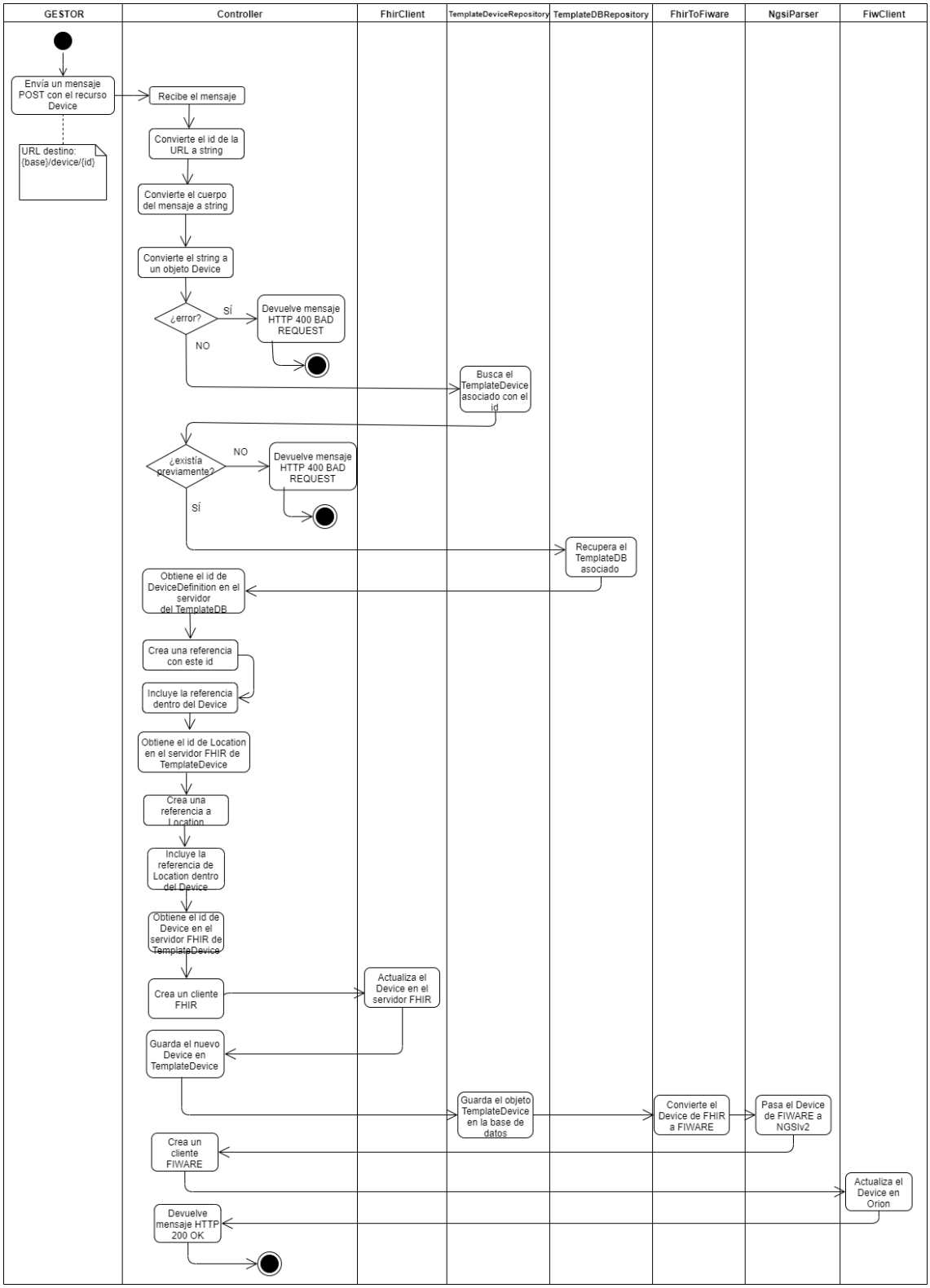

Figura 91. Diagrama de actividad actualización plantilla Device

Mediante la etiqueta @PutMapping se indica que se espera recibir un mensaje PUT, se indica también la URL ("/device/{id}") y que el cuerpo del mensaje puede estar codificado en JSON o como texto plano. Con la etiqueta @PathVariable se obtiene el identificador de la URL y se convierte a un String. Además, usando la etiqueta @RequestBody se indica que el recurso Device codificado en el cuerpo del mensaje recibido tiene que ser convertido en un String.

```
@PutMapping(value = "/device/{id}",
        consumes = {
                MediaType.APPLICATION_JSON_VALUE,
                MediaType.TEXT_PLAIN_VALUE
        })
public ResponseEntity<String> DeviceUpdate(@RequestBody String template, @PathVariable("id") String id){
```

Figura 92. Etiquetas Spring actualización Device

Lo primero que se hace es convertir el texto plano o el JSON a un objeto de tipo Device. Se debe usar un objeto del tipo IParser de la librería HAPI FHIR, se obtiene del objeto de contexto FHIR.

Se utiliza un bloque try/catch para capturar los posibles errores del recurso enviado. En caso de tener errores de sintaxis, se enviaría un mensaje HTTP 400 BAD REQUEST al usuario con un mensaje indicando que el recurso enviado no es correcto.

A continuación, hay que comprobar que el recurso enviado exista en el sistema. En caso de no haber sido creado anteriormente, se envía al usuario un mensaje HTTP 400 BAD REQUEST con un mensaje indicando el error.

En el caso de que exista el Device en el sistema, se recupera de la base de datos el identificador del recurso DeviceDefinition asociado en el servidor y se crea una referencia a este dentro del recurso Device enviado por el usuario.

Se hace algo parecido a lo anterior pero ahora con el recurso Location, se recupera su referencia y se le añade al recurso Device. Ahora, se actualiza el recurso Device en el servidor FHIR. Si todo va bien, se actualiza también el recurso en mi base de datos. Si hubiera algún error a la hora de actualizar el recurso, se mandaría un mensaje de error al usuario.

Finalmente hay que crear un recurso equivalente Device en el mundo FIWARE. Mediante la clase FhirToFiware y el recurso Device de FHIR enviado por el usuario se genera este modelo equivalente FIWARE. Para terminar, se pasa este recurso al estándar NGSIv2, se crea un cliente FIWARE y se actualiza el recurso en el Context Broker. Si todo va bien, se responde al usuario con un mensaje HTTP 200 OK y un breve mensaje de éxito.

3.8.12 Actualización de plantilla del recurso DeviceMetric

Este método se usa para actualizar una plantilla de recurso DeviceMetric previamente creada en nuestro sistema. El gestor envía por la línea de comandos, o por una aplicación como Postman, un mensaje PUT indicando el identificador de la familia del recurso a actualizar en la propia URL. La URL es de la forma “{base}/devicemetric/{id}”. El nuevo recurso DeviceMetric se envía en el cuerpo del mensaje, ya sea en formato texto plano o JSON.

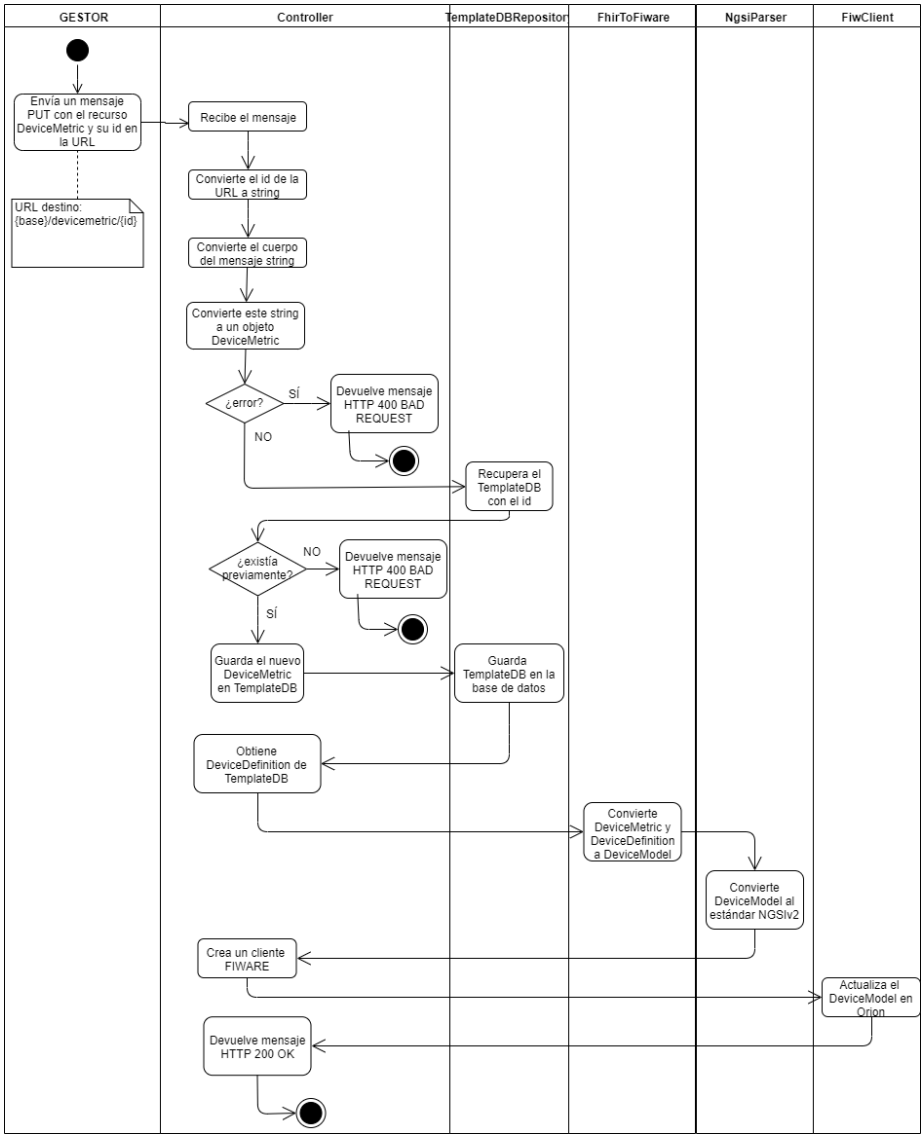

Figura 93. Diagrama de actividad actualización plantilla DeviceMetric

Mediante la etiqueta @PutMapping se indica que se espera recibir un mensaje PUT, se indica también la URL ("/devicemetric/{id}") y que el cuerpo del mensaje puede estar codificado en JSON o como texto plano. Con la etiqueta @PathVariable se obtiene el identificador de la URL y se convierte a un String. Además, usando la etiqueta @RequestBody se indica que el recurso codificado en el cuerpo del mensaje recibido tiene que ser convertido en un String.

```
@PutMapping(value = "/devicemetric/{id}",
        consumes = {
                MediaType.APPLICATION_JSON_VALUE,
                MediaType.TEXT_PLAIN_VALUE
        })
public ResponseEntity<String> DeviceMetricUpdate(@RequestBody String template, @PathVariable("id") String id){
```

Figura 94. Etiquetas Spring actualización DeviceMetric

Lo primero que se hace es convertir el texto plano o el JSON a un objeto de tipo DeviceMetric. Se debe usar un objeto del tipo IParser de la librería HAPI FHIR, se obtiene del objeto de contexto FHIR.

Se utiliza un bloque try/catch para capturar los posibles errores del recurso enviado. En caso de tener errores de sintaxis, se enviaría un mensaje HTTP 400 BAD REQUEST al usuario con un mensaje indicando que el recurso enviado no es correcto.

A continuación, hay que comprobar que el recurso enviado exista en el sistema. En caso de no haber sido creado anteriormente, se envía al usuario un mensaje HTTP 400 BAD REQUEST con un mensaje indicando el error.

En el caso de que exista en el sistema, se guarda el nuevo recurso DeviceMetric en la base de datos. Ahora hay que crear un recurso equivalente en el mundo FIWARE llamado DeviceModel. Para ello se recupera primero el recurso DeviceDefinition de la base de datos. Con este recurso recién recuperado y el recurso DeviceMetric que mandó el usuario se genera el recurso DeviceModel. Para terminar, se pasa este recurso al estándar NGSIv2, se crea un cliente FIWARE y se actualiza el recurso en el Context Broker. Si todo va bien, se responde al usuario con un mensaje HTTP 200 OK y un breve mensaje de éxito. Destacar que esta no es la vía correcta para cambiar la fecha de calibración de ningún dispositivo, sino que debe modificarse a través del Orion Context Broker. Es decir, esta funcionalidad está pensada para cambiar propiedades estáticas y no propiedades dinámicas que se actualizan a través del Orion Context Broker.

3.8.13 Actualización de plantilla del recurso Observation

Este método se usa para actualizar una plantilla de recurso Observation previamente creada en nuestro sistema. El gestor envía por la línea de comandos, o por una aplicación como Postman, un mensaje PUT indicando el identificador de la familia del recurso a actualizar en la propia URL. La URL es de la forma "{base}/observation/{id}". El nuevo recurso Observation se envía en el cuerpo del mensaje, ya sea en formato texto plano o JSON.

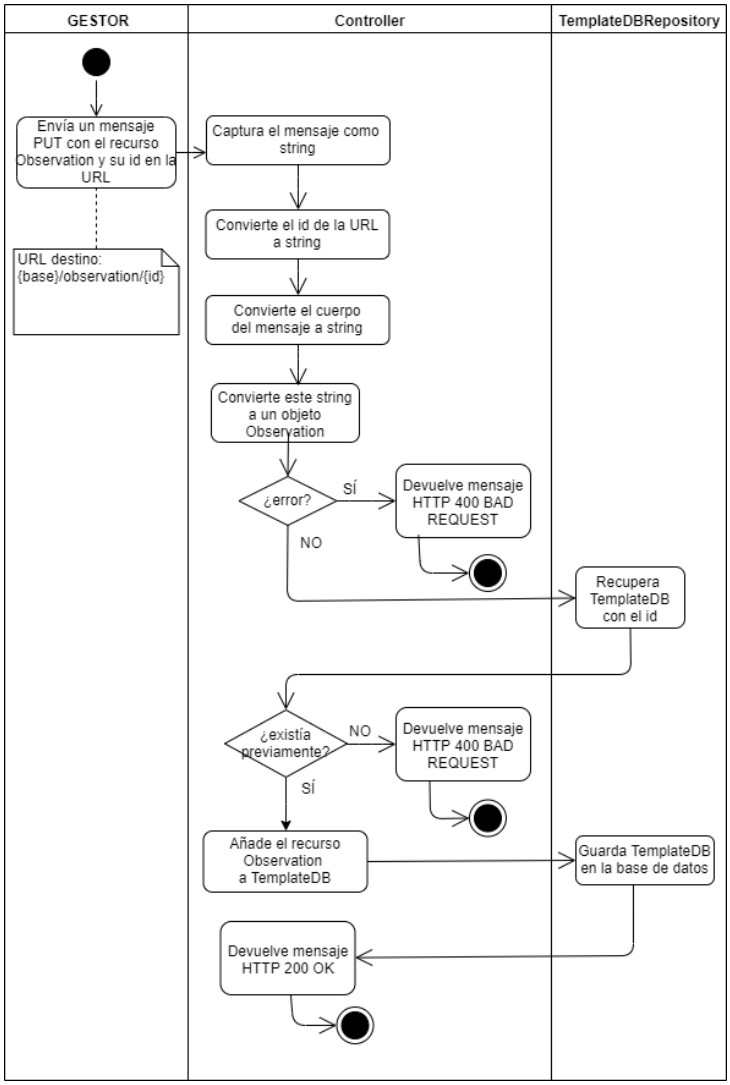

Figura 95. Diagrama de actividad actualización plantilla Observation

Mediante la etiqueta @PutMapping se indica que se espera recibir un mensaje PUT, se indica también la URL ("/observation/{id}") y que el cuerpo del mensaje puede estar codificado en JSON o como texto plano. Con la etiqueta @PathVariable se obtiene el identificador de la URL y se convierte a un String. Además, usando la etiqueta @RequestBody se indica que el recurso codificado en el cuerpo del mensaje recibido tiene que ser convertido en un String.

```
@PutMapping(value = "/observation/{id}",
        consumes = {
                MediaType.APPLICATION_JSON_VALUE
        })
public ResponseEntity<String> observationUpdate(@RequestBody String template, @PathVariable("id") String id){
```

Figura 96. Etiquetas Spring actualización Observation

Lo primero que se hace es convertir el texto plano o el JSON a un objeto de tipo Observation. Hay que usar un objeto del tipo IParser de la librería HAPI FHIR., se obtiene del objeto de contexto FHIR.

Se utiliza un bloque try/catch para capturar los posibles errores del recurso enviado. En caso de tener errores de sintaxis, se enviaría un mensaje HTTP 400 BAD REQUEST al usuario con un mensaje indicando que el recurso enviado no es correcto.

A continuación, hay que comprobar que el recurso enviado exista en el sistema. En caso de no haber sido creado anteriormente, se envía al usuario un mensaje HTTP 400 BAD REQUEST con un mensaje indicando el error.

En el caso de que exista en el sistema, se guarda la nueva plantilla de Observation en la base de datos para su posterior uso y se responde al usuario con un mensaje HTPP 200 OK, con un breve mensaje de éxito.

3.8.14 Eliminación de una familia de dispositivos

Este método se usa para borrar las entradas de la base de datos y las entidades correspondientes en FIWARE pertenecientes a una familia de dispositivos. El gestor envía por la línea de comandos, o por una aplicación como Postman, un mensaje DELETE indicando el identificador de la familia de dispositivos a borrar en la propia URL. La URL destino es del tipo "{base}/devicefamily/{id}".

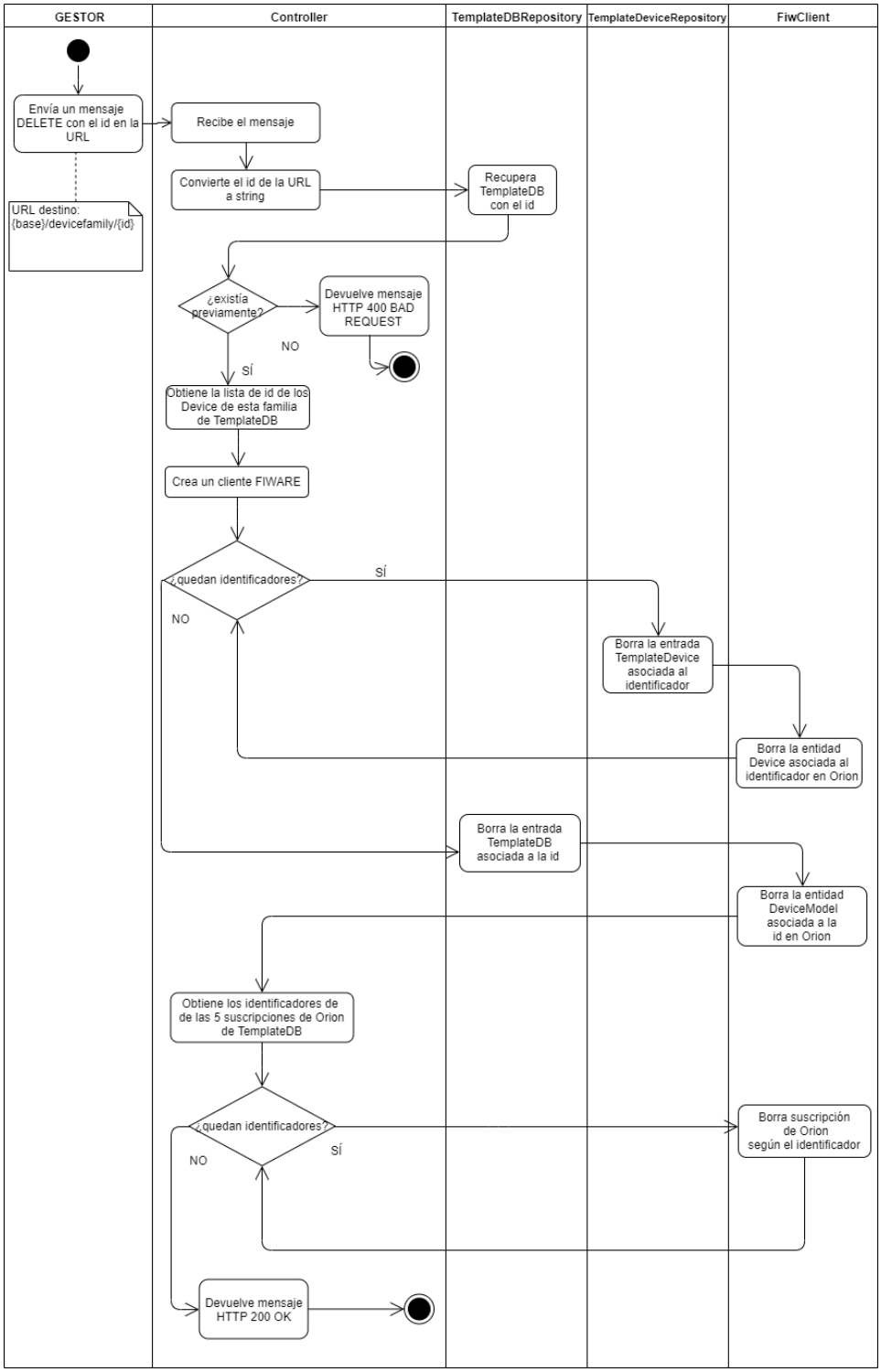

Figura 97. Diagrama de actividad eliminación familia de dispositivos

Mediante la etiqueta general @RequestMapping y seleccionando el método "RequestMethod.DELETE" se indica que se espera recibir un mensaje HTTP DELETE, se indica también la URL que es de la forma ("/devicefamily/{id}"). Donde {id} se sustituye por el identificador de la familia de dispositivos que se desea borrar. Con la etiqueta @PathVariable se obtiene el identificador de la URL y se convierte a un String.

```
@RequestMapping(value = "/devicefamily/{id}",
            method = RequestMethod.DELETE)
public ResponseEntity<String> removeDeviceFamily(@PathVariable("id") String baseId) {
```

Figura 98. Etiquetas Spring eliminación familia de dispositivos

En primer lugar, hay que recuperar de la base de datos la lista de los dispositivos registrados en la familia que se desea borrar. En caso de no existir ninguna entrada con el identificador seleccionado, se responde al usuario con un mensaje HTTP 400 Bad Request con un breve texto de error en el cuerpo. Si existiera, se recorrería la lista de dispositivos y se irían borrando uno a uno tanto en FIWARE como en la base de datos.

Finalmente, se borra la primera entrada de la familia en la base de datos y el recurso DeviceModel en FIWARE. Además, se borran las suscripciones asociadas. Se responde al usuario con un mensaje HTTP OK y un breve mensaje de éxito.

3.8.15 Eliminación de un recurso Device concreto

Este método se usa para borrar la entrada de la base de datos y la entidad correspondiente en FIWARE asociadas a un dispositivo concreto. El gestor envía por la línea de comandos, o por una aplicación como Postman, un mensaje DELETE indicando el identificador del dispositivo a borrar en la propia URL. La URL es de la forma, "{base}/device/{id}".

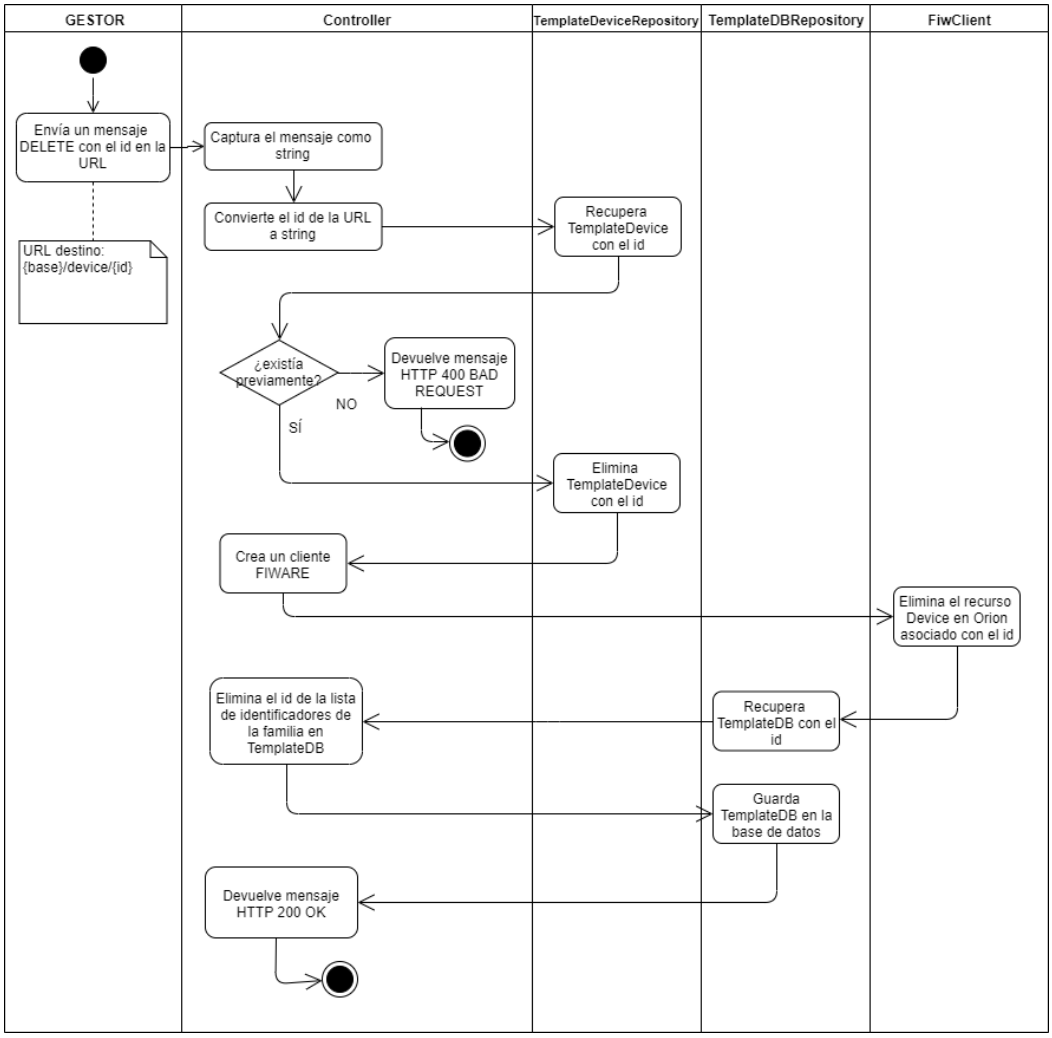

Figura 99. Diagrama de actividad eliminación Device

Mediante la etiqueta general @RequestMapping y seleccionando el método "RequestMethod.DELETE" se indica que se espera un mensaje HTTP DELETE, se indica también la URL que es de la forma ("/device/{id}"). Con la etiqueta @PathVariable se obtiene el identificador de la URL y se convierte a un String.

```
@RequestMapping(value = "/device/{id}",
        method = RequestMethod.DELETE)
public ResponseEntity<String> removeDevice(@PathVariable("id") String id) {
```

Figura 100. Etiquetas Spring eliminación Device

En primer lugar, hay que comprobar que efectivamente el Device a borrar existe en el sistema. En caso de no existir ninguna entrada con el identificador seleccionado, se responde al usuario con un mensaje HTTP 400 Bad Request con un breve texto de error en el cuerpo. En caso de existir, se borran tanto la entrada de la base de datos asociada como el recurso Device en FIWARE.

Finalmente, se borra el Device recién eliminado de la lista de dispositivos asociados a su familia. Se responde al usuario con un mensaje HTTP OK y un breve mensaje de éxito.

3.8.16 Actualización del valor medido por un dispositivo

Cuando el valor medido por una entidad en el Orion Context Broker de FIWARE se actualiza, al suscribirse correctamente a este tipo de cambio, Orion notifica a la pasarela mediante un mensaje HTTP POST con los datos pertinentes. En este caso envía el identificador del recurso afectado "id" y el nuevo valor del campo "value" a la URL "{base}/value". Recordar que tanto el destino, como el contenido del mensaje se especificaron a la hora de crear la suscripción.

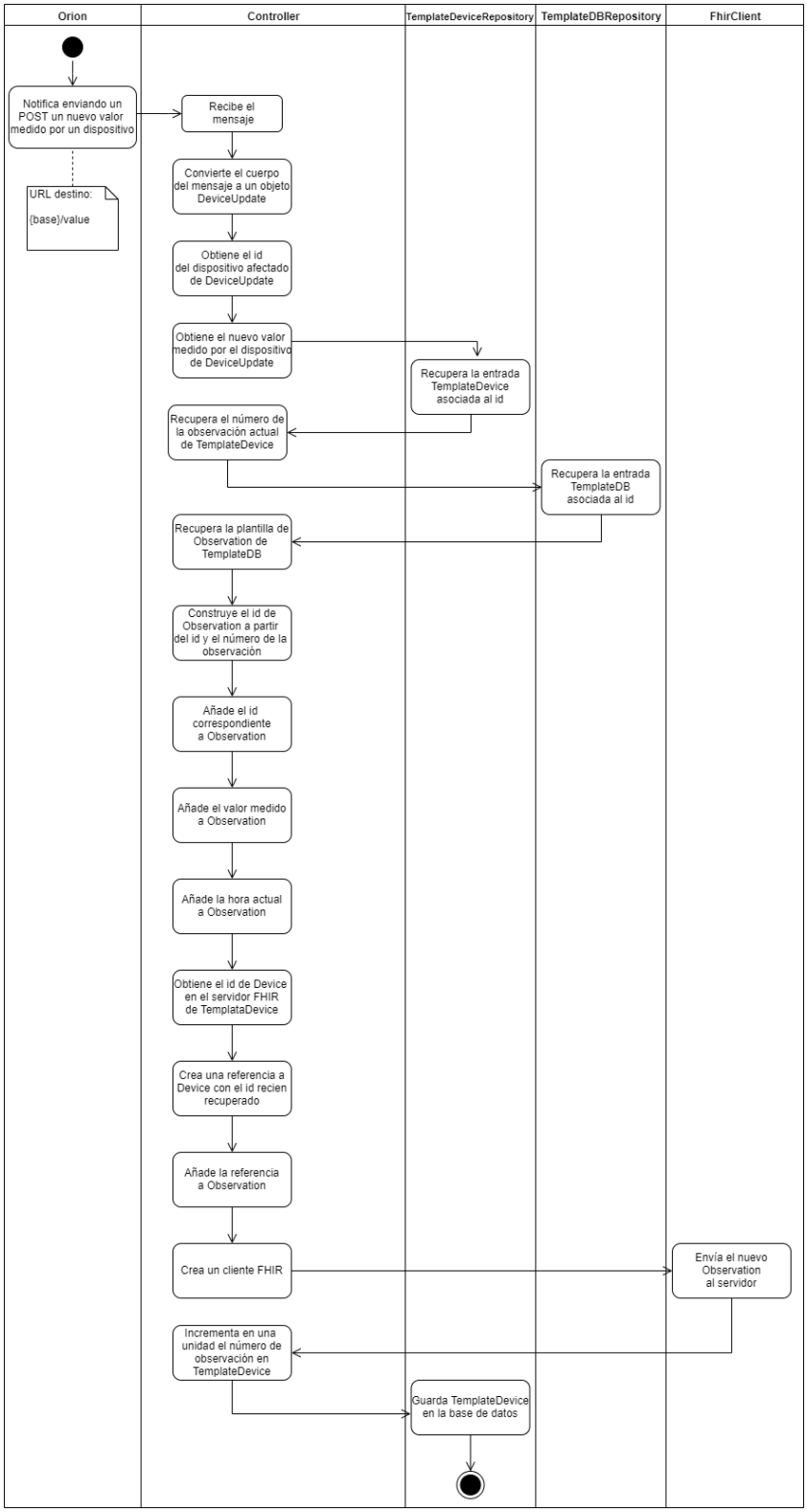

Figura 101. Diagrama de actividad actualización valor medido

Cuando el valor medido por una entidad en el Orion Context Broker de FIWARE se actualiza, al suscribirse correctamente a este tipo de cambio, Orion notifica a la pasarela mediante un mensaje HTTP POST con los datos pertinentes. En este caso envía el identificador del recurso afectado "id" y el nuevo valor del campo "value" a la URL "{base}/value". Recordar que tanto el destino, como el contenido del mensaje se especificaron a la hora de crear la suscripción.

Mediante la etiqueta @PostMapping se indica que se espera recibir un mensaje POST, se indica también la URL ("/value") y que se espera que el cuerpo del mensaje esté codificado en JSON. Además, usando la etiqueta @RequestBody se indica que el recurso JSON del cuerpo del mensaje recibido tiene que ser convertido en un objeto java de la clase DeviceUpdate. Esto ocurre de forma automática gracias a Spring y la librería Jackson.

```
@PostMapping(value = "/value",
        consumes = {
                MediaType.APPLICATION_JSON_UTF8_VALUE
        })
public ResponseEntity<HttpStatus> valueUpdate(@RequestBody DeviceUpdate update){
```

Figura 102. Etiquetas Spring actualización valor medido

Entonces, una vez recibido el POST con los datos del cuerpo del mensaje se crea un objeto DeviceUpdate en memoria. Gracias a los métodos getId() y getValue(), se recuperan el identificador de la entidad y su nuevo valor que serán de utilidad a continuación.

A continuación, usando el identificador recién recuperado se obtiene de la base de datos MongoDB el número correspondiente a esta observación. Un ejemplo de identificador sería "fiware_glucometer_1". El truncar este identificador a "fiware_glucometer", se obtiene el identificador de la familia de dispositivos. Se usa este último para recuperar de la base de datos la plantilla del recurso Observation.

A esta plantilla genérica de Observation, se le añade el identificador según el id de la entidad y el número de la observación. Si fuera el número 10, su identificador sería "fiware_glucometer_1_10".

Finalmente, se añade a la plantilla de Observation el valor que se recibió en la notificación POST y la hora actual para indicar cuando se tomó la medida. Además, hay que añadir la referencia al recurso Device al que corresponde la observación.

Ahora que está el recurso completo, se envía al servidor FHIR. Se crea un cliente Restful FHIR y acto seguido se envía la observación. Para terminar, hay que incrementar el contador que indica el número de la observación y hay guardarlo de nuevo en la base de datos. No hay que devolver nada al Orion Context Broker, ya que se trata de una notificación asíncrona.

3.8.17 Actualización del estado de un dispositivo

Cuando el estado de una entidad en el Orion Context Broker de FIWARE se actualiza, al estar suscrito correctamente a este tipo de cambio, Orion notifica a la pasarela mediante un mensaje HTTP POST con los datos pertinentes. En este caso envía el identificador del recurso "id" y el nuevo valor del campo "deviceState" a la URL "{base}/value" de la pasarela. El destino y el contenido del mensaje se especificaron a la hora crear la suscripción.

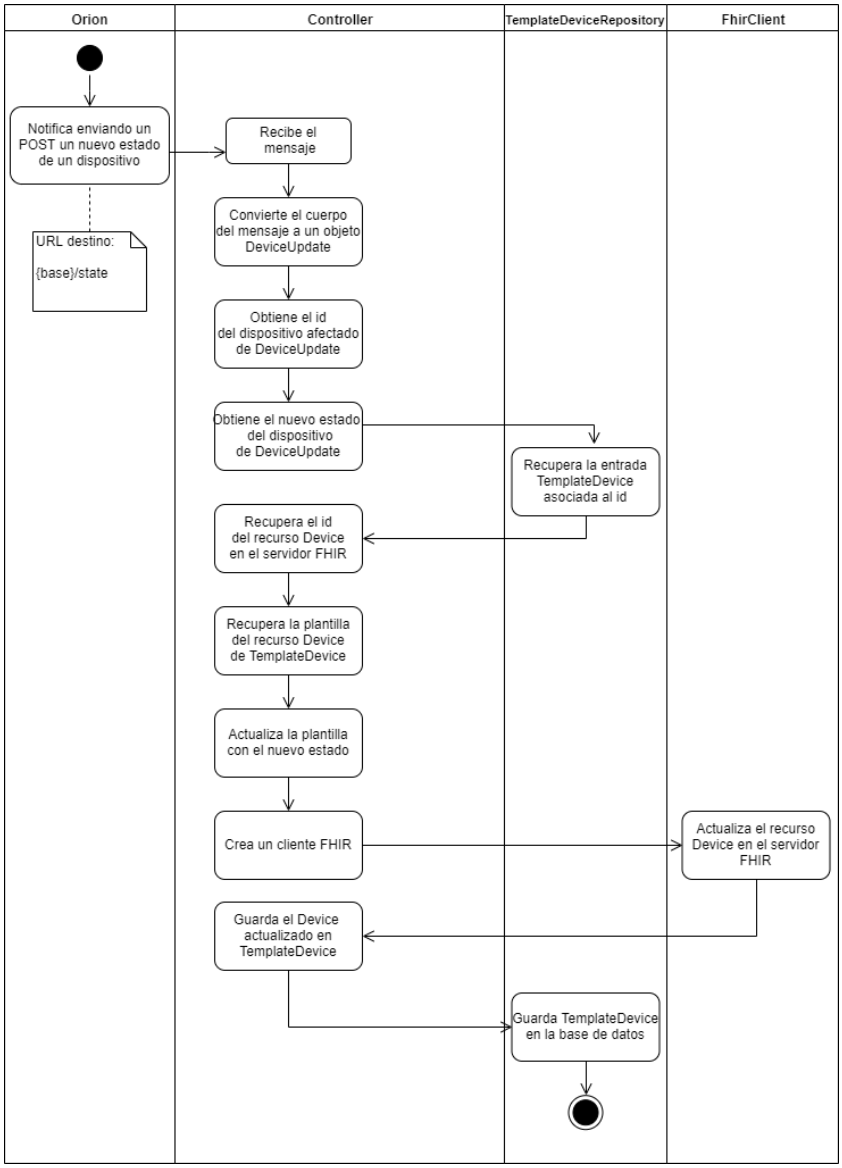

Figura 103. Diagrama de actividad actualización del estado

Mediante la etiqueta @PostMapping se indica que esperamos un mensaje POST, se indica también la url ("/state") y que se espera que el cuerpo del mensaje esté codificado en JSON. Además, usando la etiqueta @RequestBody se indica que el recurso JSON del cuerpo del mensaje recibido tiene que convertirse en un objeto java de la clase DeviceUpdate.

```
@PostMapping(value = "/state",
        consumes = {
                MediaType.APPLICATION_JSON_VALUE
        })
public ResponseEntity<HttpStatus> stateUpdate(@RequestBody DeviceUpdate update){
```

Figura 104. Etiquetas Spring actualización del estado

Entonces, una vez recibido el mensaje POST con los datos del cuerpo se crea un objeto DeviceUpdate en memoria. Gracias a los métodos getId() y getState(), se recuperan el identificador de la entidad y su nuevo estado que serán de utilidad a continuación.

A continuación, se usa este identificador para obtener de la base de datos la plantilla del recurso Device y su identificador del servidor FHIR para poder actualizar el recurso posteriormente.

Hay que actualizar la plantilla con uno de los 4 valores posibles, “active”, “inactive”, “enteredinerror” y “unkown”. Finalmente, se actualiza el recurso en el servidor FHIR y se guarda de nuevo en mi base de datos.

3.8.18 Actualización de la posición de un dispositivo

Cuando la posición de una entidad en el Orion Context Broker de FIWARE se actualiza, al estar suscrito correctamente a este tipo de cambio, Orion notifica a la pasarela mediante un mensaje HTTP POST con los datos pertinentes. En este caso envía el identificador del recurso “id” y el nuevo valor del campo “location” a la URL “{base}/location”. Recordar que tanto el destino, como el contenido del mensaje se especificaron a la hora de crear la suscripción.

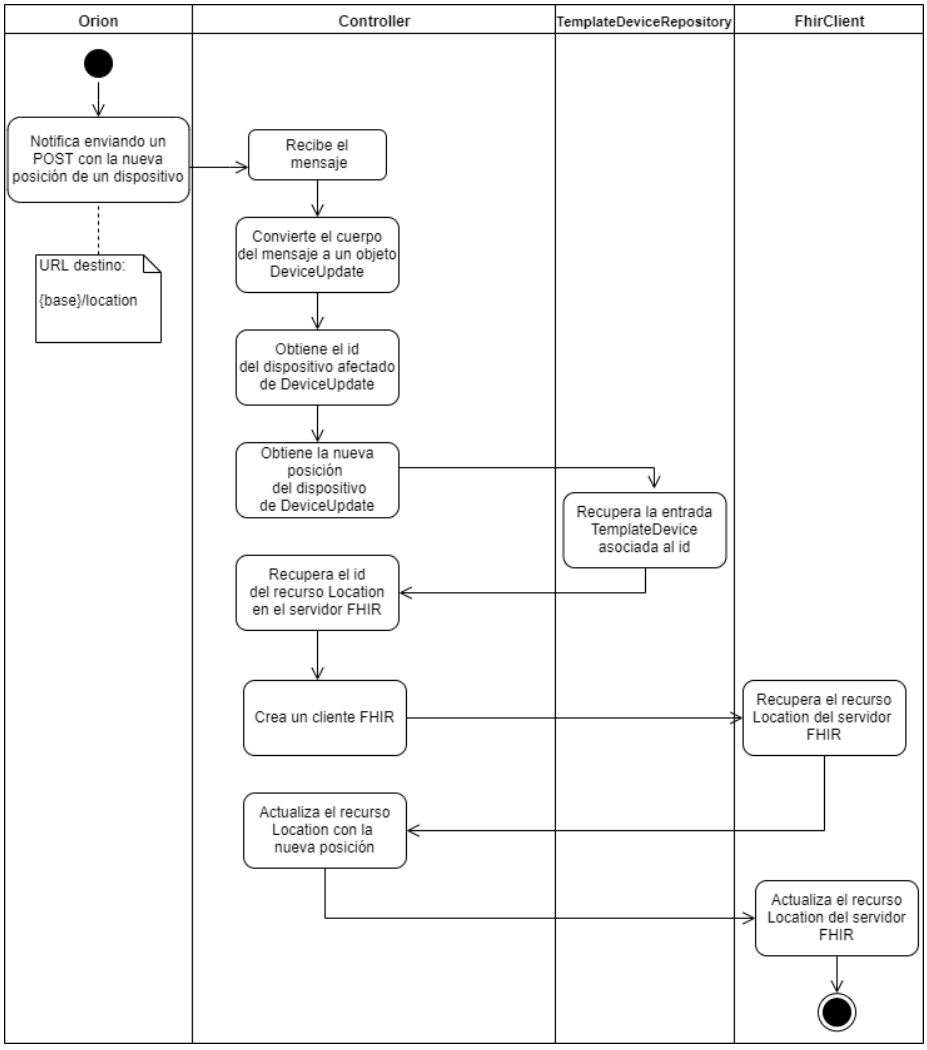

Figura 105. Diagrama de actividad actualización de la posición

Mediante la etiqueta @PostMapping se indica que esperamos recibir un mensaje POST, se indica también la URL (“/location”) y que se espera que el cuerpo del mensaje esté codificado en JSON.

Además, se usa la etiqueta @RequestBody para indicar que el recurso JSON del cuerpo del mensaje recibido tiene que ser convertido en un objeto java de la clase DeviceUpdate.

```
@PostMapping(value = "/location",
        consumes = {
                MediaType.APPLICATION_JSON_VALUE
        })
public ResponseEntity<HttpStatus> locationUpdate(@RequestBody DeviceUpdate update){
```

Figura 106. Etiquetas Spring actualización localización

Entonces, una vez recibido el mensaje POST con los datos del cuerpo se crea un objeto DeviceUpdate en memoria. Gracias a los métodos getId() y getLocation(), se recuperan el identificador de la entidad y su nueva localización que serán de utilidad a continuación.

A continuación, usando el identificador recién recuperado, se obtiene de la base de datos el identificador del recurso Location en el servidor FHIR.

Se crea el cliente FHIR y se recupera el recurso del servidor para actualizarlo. Finalmente, se actualiza la posición y se envía el recurso actualizado al servidor FHIR.

3.8.19 Actualización de la fecha de calibración de un dispositivo

Cuando la fecha de calibración de una entidad en el Orion Context Broker de FIWARE se actualiza, al estar suscrito correctamente a este tipo de cambio, Orion notifica a la pasarela mediante un mensaje HTTP POST con los datos pertinentes. En este caso envía el identificador del recurso “id” y el nuevo valor del campo “dateLastCalibration” a la URL “{base}/value”. Tanto el contenido del mensaje como el destino se especificaron a la hora de crear la suscripción.

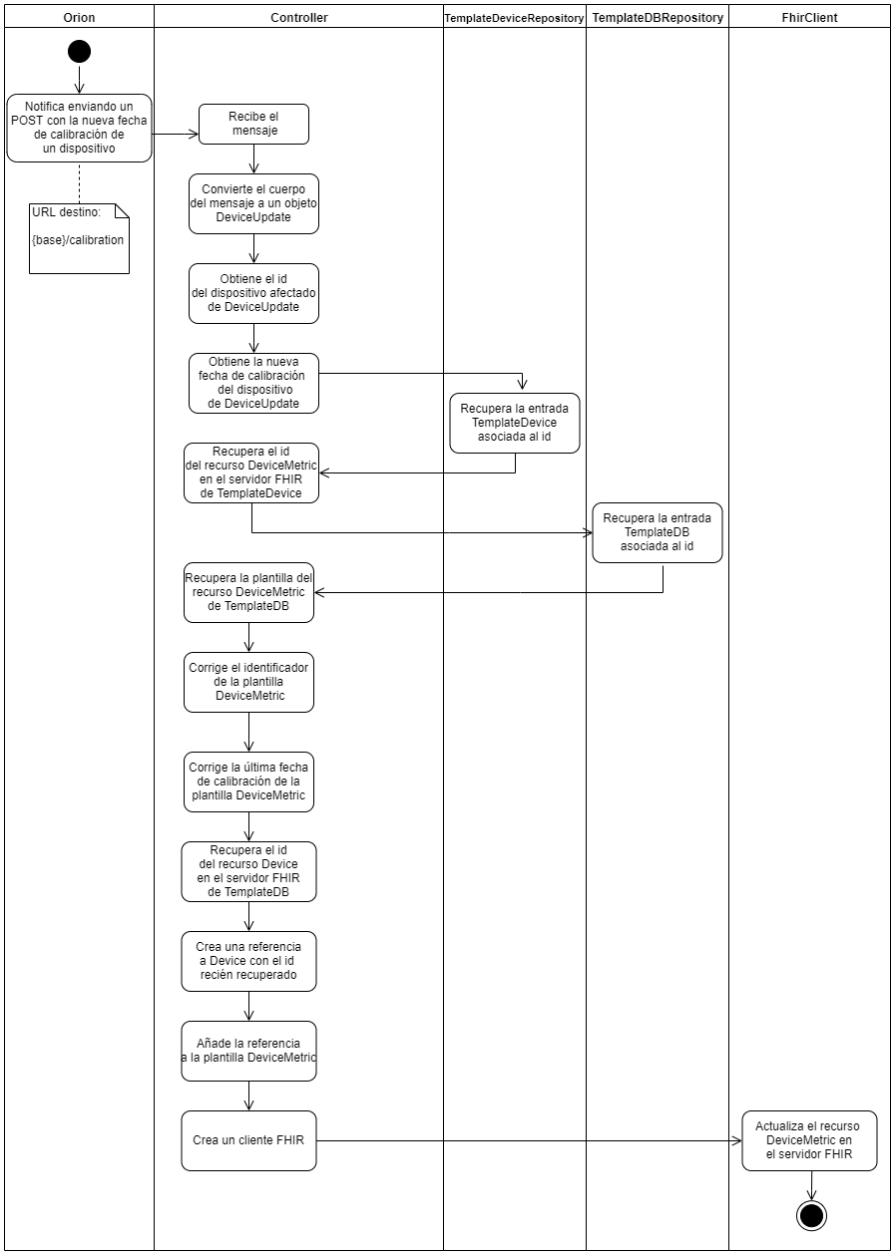

Figura 107. Diagrama de actividad actualización de la fecha de calibración

Mediante la etiqueta @PostMapping se indica que esperamos recibir un mensaje POST, se indica también la URL ("/calibration") y que se espera que el cuerpo del mensaje esté codificado en JSON. Además, usando la etiqueta @RequestBody el recurso JSON del cuerpo del mensaje recibido tiene que convertise en un objeto java de la clase DeviceUpdate.

```
@PostMapping(value = "/calibration",
        consumes = {
                MediaType.APPLICATION_JSON_VALUE
        })
public void calibrationUpdate(@RequestBody DeviceUpdate update){
```

Figura 108. Etiquetas Spring actualización fecha de calibración

Entonces, una vez recibido el mensaje POST con los datos del cuerpo se crea un objeto DeviceUpdate en memoria. Gracias a los métodos getId() y getDateLastCalibration(), se recupera el identificador de la entidad y su última fecha de calibración que serán de utilidad a continuación.

A continuación, usando el identificador recién recuperado, se obtiene de la base de datos el identificador del recurso DeviceMetric en el servidor FHIR. Como se ha comentado con anterioridad, al truncar el identificador del dispositivo, se obtiene el identificador de la familia de dispositivos. Se usa este último para recuperar la plantilla del recurso DeviceMetric de la base de datos.

Ahora, se modifican el identificador y la última fecha de calibración de esta plantilla. Se añade la referencia al recurso Device al que corresponde el DeviceMetric. Finalmente, se actualiza el recurso en el servidor FHIR.

3.8.20 Actualización de la dirección IP de un dispositivo

Cuando la dirección IP de una entidad en el Orion Context Broker de FIWARE se actualiza, al estar suscrito correctamente a este tipo de cambio, Orion notifica a la pasarela mediante un mensaje HTTP POST con los datos pertinentes. En este caso envía el identificador del recurso "id" y el nuevo valor del campo "ipAddress" a la URL "{base}/ipaddress".

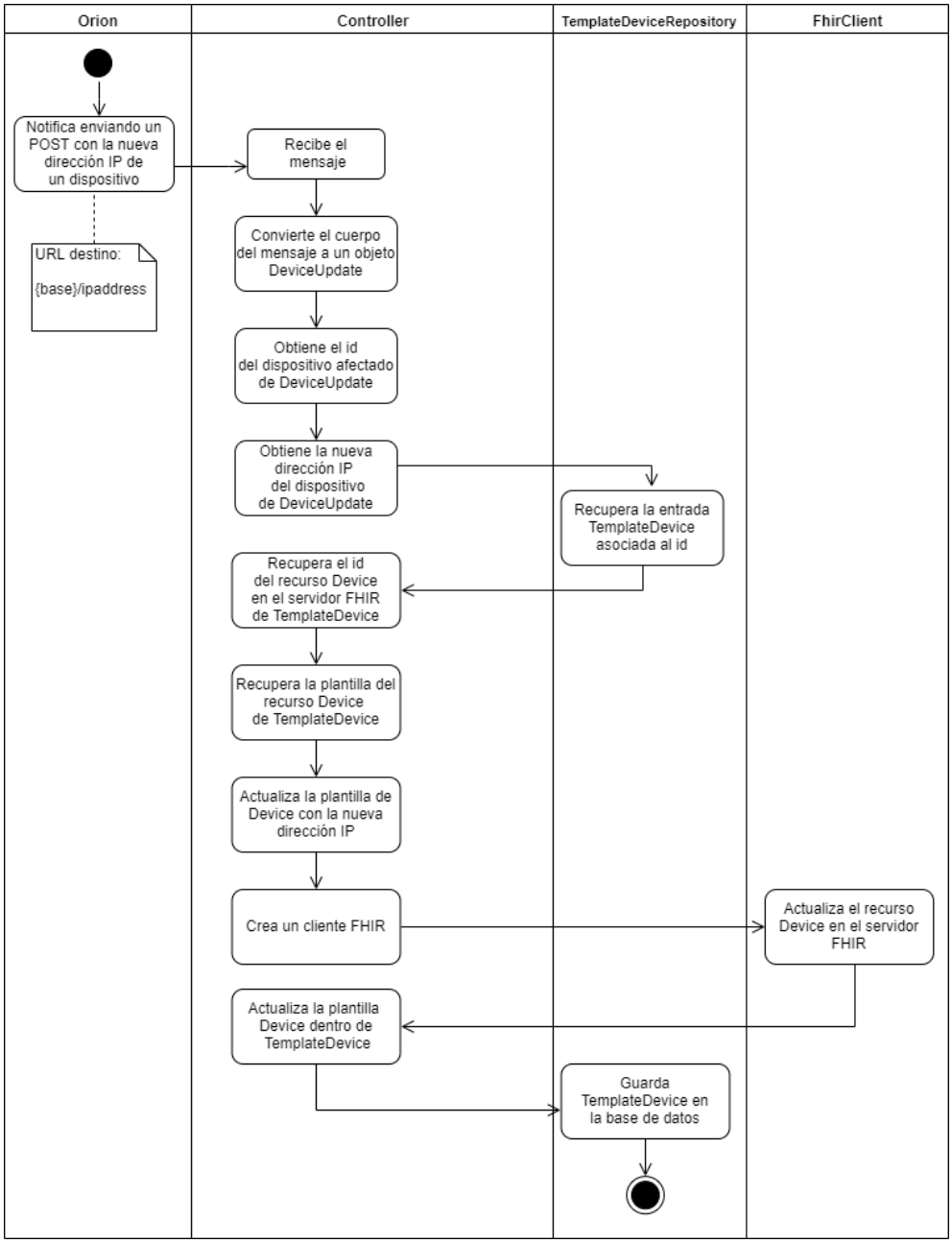

Figura 109. Diagrama de actividad actualización de la IP

Mediante la etiqueta @PostMapping se indica que se espera recibir un mensaje POST, se indica también la URL ("/ipaddress") y que el cuerpo del mensaje tiene q estar codificado en JSON. Además, usando la etiqueta @RequestBody el recurso JSON del cuerpo del mensaje recibido se convierte en un objeto java de la clase DeviceUpdate.

```
@PostMapping(value = "/ipaddress",
        consumes = {
                MediaType.APPLICATION_JSON_VALUE
        })
public void ipAddressUpdate(@RequestBody DeviceUpdate update){
```

Figura 110. Etiquetas Spring actualización dirección IP

Entonces, una vez recibido el mensaje POST con los datos del cuerpo se crea un objeto DeviceUpdate en memoria. Gracias a los métodos getId() y getIpAddress(), se recupera el

identificador de la entidad y la nueva dirección IP de la entidad que serán de utilidad a continuación.

A continuación, usando el identificador recién recuperado, se obtiene de la base de datos la plantilla del recurso Device, así como el identificador del recurso en el servidor FHIR.

Ahora, se actualiza la plantilla obtenida con la nueva IP, se crea un cliente FHIR y se sube el recurso al servidor. Finalmente, se guarda la plantilla actualizada en la base de datos para su posterior uso.

Hay que guardarla porque cuando se actualiza el estado de la entidad, también se accede a esta plantilla del recurso Device. Sino se guardase se perdería el cambio de IP al actualizar el estado del dispositivo.

4 MONTAJE Y PRUEBAS DEL ENTORNO

4.1 Servidor FHIR

Cuando se empezó a desarrollar la aplicación se usaba un servidor FHIR público, donde cualquiera podía subir y modificar recursos. No obstante, los servidores de este tipo algunas veces no estaban disponibles cuando hacían falta. Además, eran purgados a menudo, borrando todos los recursos creados por la aplicación. Por lo tanto, para la implementación final se decidió lanzar un servidor local con ayuda de la librería HAPI FHIR. Esta librería ofrece una implementación del servidor equivalente a las que usan los mencionados servidores públicos.

4.1.1 Prerrequisitos

En primer lugar, hay que clonar este proyecto de Github: https://github.com/hapifhir/hapi-fhir-jpaserver-starter . Otra opción sería hacer un "fork" para poder realizar modificaciones y subirlo de nuevo a la plataforma Github.

Este es el proyecto recomendado por la documentación de HAPI FHIR para lanzar un servidor propio, funciona exactamente igual que el servidor de pruebas antes comentado.

Por lo tanto, se instala la herramienta git por la línea de comandos para clonar el proyecto del servidor FHIR.

```
jaimegruiz@ubuntu:~/Desktop$ sudo apt install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
```

Figura 111. Instalación git

```
jaimegruiz@ubuntu:~/Desktop$ git clone https://github.com/hapifhir/hapi-fhir-jpa
server-starter
Cloning into 'hapi-fhir-jpaserver-starter'...
remote: Enumerating objects: 59, done.
remote: Counting objects: 100% (59/59), done.
remote: Compressing objects: 100% (37/37), done.
remote: Total 994 (delta 15), reused 50 (delta 11), pack-reused 935
Receiving objects: 100% (994/994), 352.69 KiB | 1.10 MiB/s, done.
Resolving deltas: 100% (328/328), done.
```

Figura 112. Git clone servidor hapifhir

Hay que tener instalado el JDK (Java Development Kit) versión 8 como mínimo. Si se introduce **"java -version"** o **"javac – versión"** en la terminal se puede averiguar si está instalado y qué versión.

```
jaimegruiz@ubuntu:~/Desktop$ java --version
java 12.0.2 2019-07-16
Java(TM) SE Runtime Environment (build 12.0.2+10)
Java HotSpot(TM) 64-Bit Server VM (build 12.0.2+10, mixed mode, sharing)
jaimegruiz@ubuntu:~/Desktop$ javac --version
javac 12.0.2
```

Figura 113.Versión java

En caso de no tenerlo instalado en el equipo se puede descargar de la página de Oracle : https://www.oracle.com/technetwork/java/javase/downloads/jdk12-downloads-5295953.html

Para Ubuntu, Linux Mint y Debian hay un instalador realmente fácil de usar. Primero se añade el siguiente repositorio.

```
jaimegruiz@ubuntu:~/Desktop$ sudo add-apt-repository ppa:linuxuprising/java
 Oracle Java 11 (LTS) and 12 installer for Ubuntu, Linux Mint and Debian.

Java binaries are not hosted in this PPA due to licensing. The packages in th
is PPA download and install Oracle Java 11, so a working Internet connection
is required.

The packages in this PPA are based on the WebUpd8 Oracle Java PPA packages: h
ttps://launchpad.net/~webupd8team/+archive/ubuntu/java
```

Figura 114.Instalador java

Después, hay que ejecutar **"sudo apt-get update"** para obtener las últimas versiones de los paquetes y sus dependencias. Finalmente, se introduce **"sudo apt-get install Oracle-java12-installer"** lo que arrancará una simple interfaz gráfica.

```
jaimegruiz@ubuntu:~/Desktop$ sudo apt-get update
Hit:1 https://download.docker.com/linux/ubuntu bionic InRelease
Hit:2 http://ppa.launchpad.net/linuxuprising/java/ubuntu bionic InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Get:4 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB
]
Fetched 252 kB in 6s (45.4 kB/s)
Reading package lists... Done
jaimegruiz@ubuntu:~/Desktop$ sudo apt-get install oracle-java12-installer
Reading package lists... Done
Building dependency tree
Reading state information... Done
oracle-java12-installer is already the newest version (12.0.2-1~linuxuprising
0).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
```

Figura 115. Actualización de paquetes y dependencias

Además, hay que instalar la última versión de la herramienta Apache Maven. En Ubuntu, simplemente hay que ejecutar **"sudo apt install maven"**

```
jaimegruiz@ubuntu:~/Desktop$ sudo apt install maven
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
```

Figura 116. Instalación Maven

4.1.2 Ejecutando el servidor

La forma más fácil de lanzar el servidor es directamente con Maven, usando el servidor Jetty instalado por defecto. Solo se necesita ejecutar el comando **"mvn jetty:run"** en el directorio principal del proyecto que se ha clonado anteriormente. Tras esto se encontrará el servidor FHIR en la URL: http://localhost:8080/hapi-fhir-jpaserver/ . Destacar que la primera vez que se ejecute este comando, Maven va a proceder a descargar todas las dependencias del proyecto por lo que requiere un tiempo mayor que las próximas ejecuciones.

```
jaimegruiz@ubuntu:~/Desktop/hapi-fhir-jpaserver-starter$ mvn jetty:run
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.google.inject.internal.cglib.core.$ReflectUtils$1 (file:/usr/share/maven/lib/
t,java.security.ProtectionDomain)
WARNING: Please consider reporting this to the maintainers of com.google.inject.internal.cglib.core.$ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< ca.uhn.hapi.fhir:hapi-fhir-jpaserver-starter >------------
[INFO] Building HAPI FHIR JPA Server - Starter Project 4.0.0
[INFO] --------------------------------[ war ]---------------------------------
[INFO]
[INFO] >>> jetty-maven-plugin:9.4.8.v20180619:run (default-cli) > test-compile @ hapi-fhir-jpaserver-starter >>>
```

Figura 117. Lanzamiento del servidor FHIR desde Maven

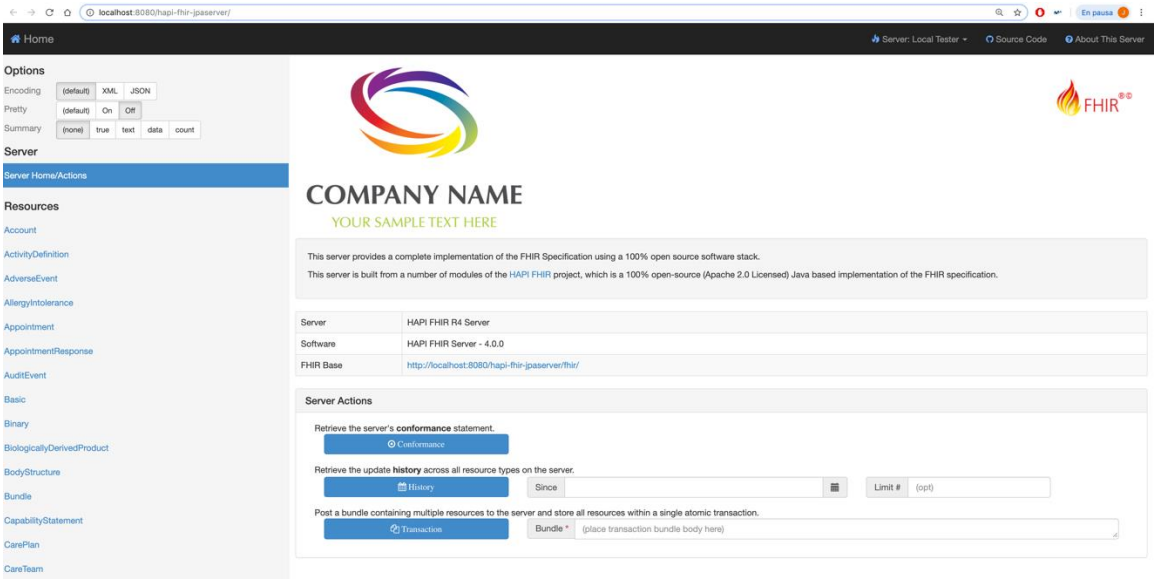

Figura 118. Página de inicio del servidor FHIR.

Como se ha comentado, esta es la forma más fácil pero no la más recomendable en un entorno real. Así que se ha dado un paso más y se ha lanzado un contenedor Apache Tomcat con la aplicación del servidor, como recomiendan en la documentación de Github de HAPI FHIR.

4.1.3 Ejecutando el servidor dentro de Apache Tomcat

Primero hay que descargar los archivos binarios comprimidos en zip Core de la página de descargas de Apache Tomcat: https://tomcat.apache.org/download-90.cgi

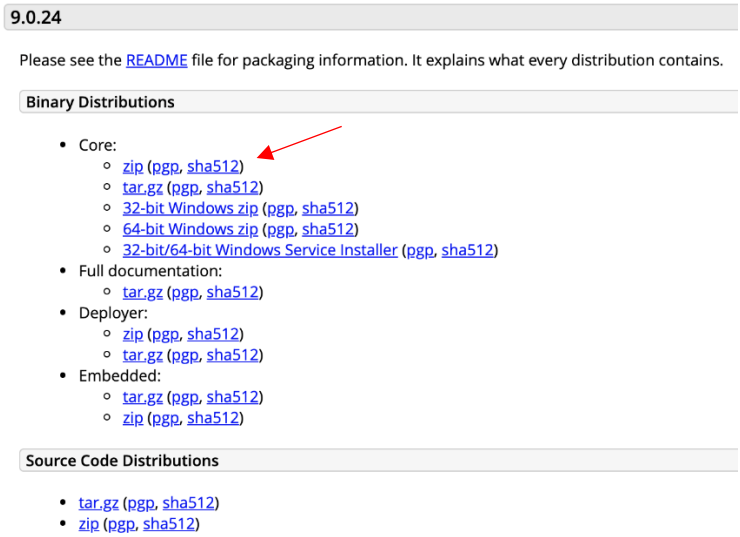

Figura 119. Página de descargas Apache Tomcat

A continuación, se descomprime el archivo, obteniendo una carpeta con un nombre parecido a apache-tomcat-9.0.24. Hay que mover esta carpeta a la dirección "/usr/local". La forma más fácil de hacerlo es ejecutando este comando en la terminal **"sudo mv ~/Downloads/apache-tomcat-9.0.24/ /usr/local/"**.

Ahora hay que hacerse dueño de esa carpeta con el comando **"sudo chown -R jaimegruiz /usr/local/apache-tomcat-9.0.24".** Cambiando jaimegruiz por tu nombre de usuario del sistema. Finalmente le damos permisos de ejecución a los scripts de Tomcat con este comando **"sudo chmod +x /usr/local/apache-tomcat-9.0.24/bin/*.sh"** .

```
jaimegruiz@ubuntu:~$ cd /home/jaimegruiz/Downloads/
jaimegruiz@ubuntu:~/Downloads$ unzip apache-tomcat-9.0.24.zip
Archive:  apache-tomcat-9.0.24.zip
   creating: apache-tomcat-9.0.24/
   creating: apache-tomcat-9.0.24/bin/
   creating: apache-tomcat-9.0.24/conf/
```

Figura 120. Descompresión Tomcat

```
jaimegruiz@ubuntu:~/Downloads$ sudo mv  ~/Downloads/apache-tomcat-9.0.24/ /usr/local/
jaimegruiz@ubuntu:~/Downloads$ ls
apache-tomcat-9.0.24.zip
jaimegruiz@ubuntu:~/Downloads$ sudo chown -R jaimegruiz /usr/local/apache-tomcat-9.0.24/
jaimegruiz@ubuntu:~/Downloads$ sudo chmod +x /usr/local/apache-tomcat-9.0.24/bin/*.sh
```

Figura 121. Permisos de ejecución Tomcat

Ya está Tomcat listo, para iniciar el servidor desde la terminal se usa el comando **"/usr/local/apache-tomcat-9.0.24/bin/startup.sh".** Si todo ha ido bien, aparecerá el mensaje "Tomcat started".

```
jaimegruiz@ubuntu:~/Downloads$ /usr/local/apache-tomcat-9.0.24/bin/startup.sh
Using CATALINA_BASE:   /usr/local/apache-tomcat-9.0.24
Using CATALINA_HOME:   /usr/local/apache-tomcat-9.0.24
Using CATALINA_TMPDIR: /usr/local/apache-tomcat-9.0.24/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /usr/local/apache-tomcat-9.0.24/bin/bootstrap.jar:/usr/local/apac
he-tomcat-9.0.24/bin/tomcat-juli.jar
Tomcat started.
```

Figura 122. Arranque de Tomcat

Para comprobar que está funcionando de verdad, basta con abrir un navegador e ir a la dirección http://localhost:8080/. Debería aparecer la página de inicio de Tomcat.

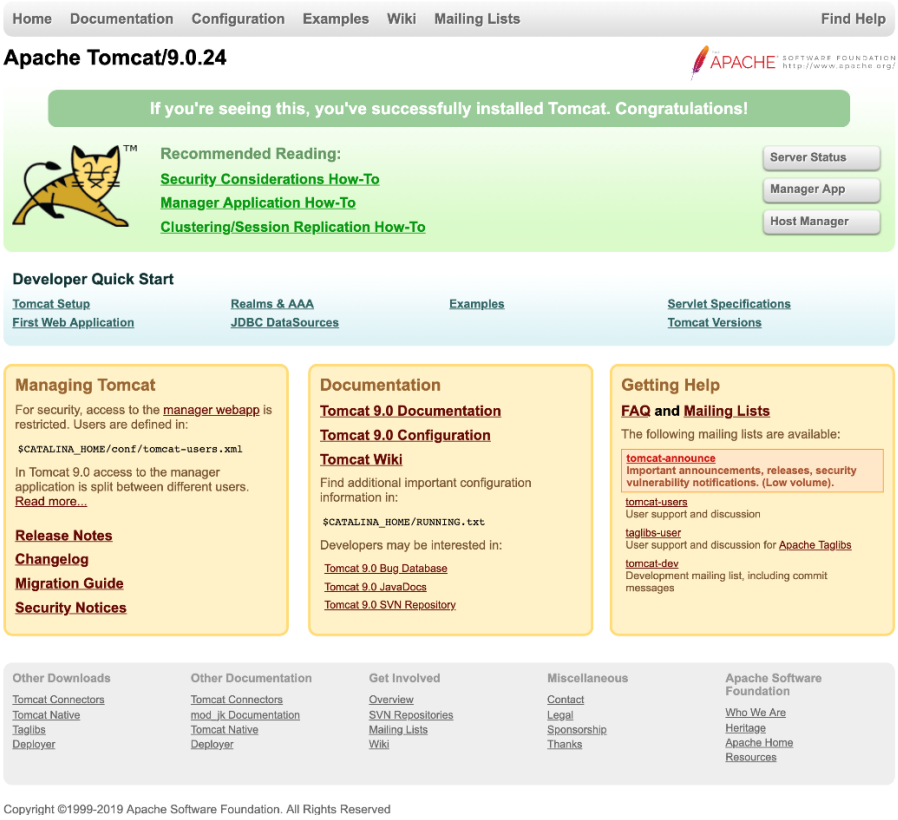

Figura 123. Página de inicio servidor Tomcat

Para detener el servidor Tomcat, se escribe el siguiente comando en la terminal: **"/usr/local/Tomcat/bin/shutdown.sh".** Efectivamente si se carga la página de nuevo ya no estará disponible.

```
jaimegruiz@ubuntu:~/Downloads$ /usr/local/apache-tomcat-9.0.24/bin/shutdown.sh
Using CATALINA_BASE:   /usr/local/apache-tomcat-9.0.24
Using CATALINA_HOME:   /usr/local/apache-tomcat-9.0.24
Using CATALINA_TMPDIR: /usr/local/apache-tomcat-9.0.24/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /usr/local/apache-tomcat-9.0.24/bin/bootstrap.jar:/usr/local/apac
he-tomcat-9.0.24/bin/tomcat-juli.jar
NOTE: Picked up JDK_JAVA_OPTIONS:  --add-opens=java.base/java.lang=ALL-UNNAMED --add-ope
ns=java.base/java.io=ALL-UNNAMED --add-opens=java.rmi/sun.rmi.transport=ALL-UNNAMED
```
Figura 124. Detención Tomcat

Ahora hay que crear un archivo WAR que contenga la aplicación del servidor FHIR. Para conseguir esto se ejecuta el comando "mvn clean install" dentro del directorio principal del proyecto clonado de Github.

```
jaimegruiz@ubuntu:~/Desktop/hapi-fhir-jpaserver-starter$ mvn clean install
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.google.inject.internal.cglib.core.$ReflectUtil
s$1 (file:/usr/share/maven/lib/guice.jar) to method java.lang.ClassLoader.defineClass(ja
va.lang.String,byte[],int,int,java.security.ProtectionDomain)
WARNING: Please consider reporting this to the maintainers of com.google.inject.internal
.cglib.core.$ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective acce
ss operations
WARNING: All illegal access operations will be denied in a future release
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< ca.uhn.hapi.fhir:hapi-fhir-jpaserver-starter >------------
[INFO] Building HAPI FHIR JPA Server - Starter Project 4.0.0
[INFO] --------------------------------[ war ]---------------------------------
Downloading from ossrh: https://oss.sonatype.org/content/repositories/snapshots/org/apac
```

```
[INFO] Installing /home/jaimegruiz/Desktop/hapi-fhir-jpaserver-starter/target/hapi-fhir-
jpaserver.war to /home/jaimegruiz/.m2/repository/ca/uhn/hapi/fhir/hapi-fhir-jpaserver-st
arter/4.0.0/hapi-fhir-jpaserver-starter-4.0.0.war
[INFO] Installing /home/jaimegruiz/Desktop/hapi-fhir-jpaserver-starter/pom.xml to /home/
jaimegruiz/.m2/repository/ca/uhn/hapi/fhir/hapi-fhir-jpaserver-starter/4.0.0/hapi-fhir-j
paserver-starter-4.0.0.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:17 min
[INFO] Finished at: 2019-09-17T09:39:59-07:00
[INFO] ------------------------------------------------------------------------
```

Figura 125. Creación archivo WAR con la aplicación del servidor FHIR

Esto creará un archivo con el nombre “hapi-fhir-jpaserver.war” en el directorio “target” situado en el directorio raíz del proyecto.

Ahora hay que mover este archivo “hapi-fhir-jpaserver.war” al directorio de Tomcat “/webapps/” y ejecutar de nuevo Tomcat con el comando **“/usr/local/Tomcat/bin/startup.sh”.**
Al ejecutarse, se generá la carpeta “hapi-fhir-jpaserver” dentro de “/webapps/” y se podrá observar el servidor FHIR en la URL: http://localhost:8080/hapi-fhir-jpaserver/ . Se ha cambiado la imagen que aparece en la página principal del servidor.

```
jaimegruiz@ubuntu:~/Desktop/hapi-fhir-jpaserver-starter$ cd target/
jaimegruiz@ubuntu:~/Desktop/hapi-fhir-jpaserver-starter/target$ ls
classes                     generated-test-sources   maven-archiver
duplicate-finder-result.xml hapi-fhir-jpaserver      maven-status
failsafe-reports            hapi-fhir-jpaserver.war  test-classes
generated-sources           lucenefiles              war
jaimegruiz@ubuntu:~/Desktop/hapi-fhir-jpaserver-starter/target$ mv hapi-fhir-jpaserver.war
 //usr/local/apache-tomcat-9.0.24/webapps/
jaimegruiz@ubuntu:~/Desktop/hapi-fhir-jpaserver-starter/target$ /usr/local/apache-tomcat-9
.0.24/bin/startup.sh
Using CATALINA_BASE:   /usr/local/apache-tomcat-9.0.24
Using CATALINA_HOME:   /usr/local/apache-tomcat-9.0.24
Using CATALINA_TMPDIR: /usr/local/apache-tomcat-9.0.24/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /usr/local/apache-tomcat-9.0.24/bin/bootstrap.jar:/usr/local/apache
-tomcat-9.0.24/bin/tomcat-juli.jar
Tomcat started.
```

Figura 126. Traslado del WAR a Tomcat

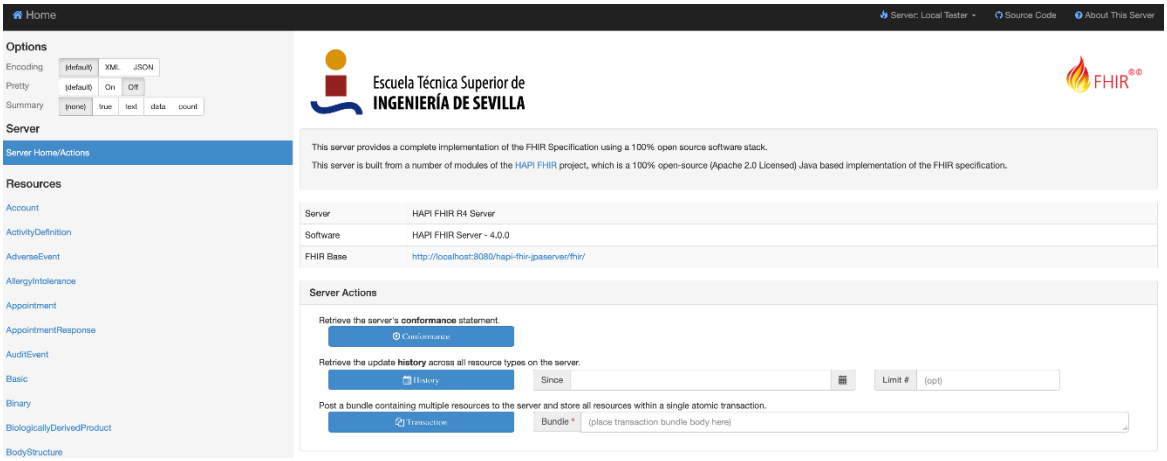

Figura 127. Página de inicio servidor FHIR (imagen ETSI)

4.1.4 Ejecutando el servidor dentro de un contenedor Docker (Opcional)

Dentro del directorio hapi-fhir-jpaserver-starter se encuentra el archivo “build-docker-image.sh” que contiene lo siguiente.

```
#!/bin/sh

mvn package && \
  docker build -t hapi-fhir/hapi-fhir-jpaserver-starter .
```

Figura 128. Archivo "build-docker-image.sh"

Esto se encarga de empaquetar la aplicación java con Maven y crea una imagen de Docker con el nombre hapi-fhir/hapi-fhir-jpaserver-starter. El punto final indica que el archivo de configuración Dockerfile se encuentra en el mismo directorio. Este Dockerfile contiene lo siguiente:

```
1  FROM jetty:9-jre8-alpine
2  USER jetty:jetty
3  ADD ./target/hapi-fhir-jpaserver.war /var/lib/jetty/webapps/hapi-fhir-jpaserver.war
4  EXPOSE 8080
```

Figura 129. Dockerfile

Esto le indica a Docker que tiene que hacer un pull de la imagen jetty:9-jre8-alpine, que es una distribución muy básica de Linux con la aplicación jetty instalada para desplegar contenedores web. También indica el nombre de usuario “jetty:jetty” y transfiere el paquete war recién creado por Maven, que contiene el servidor FHIR, a la carpeta webapps de jetty para ser desplegado. Finalmente abre el puerto 8080 al exterior.

Por lo tanto, hay que ejecutar el archivo “build-docker-image.sh”.

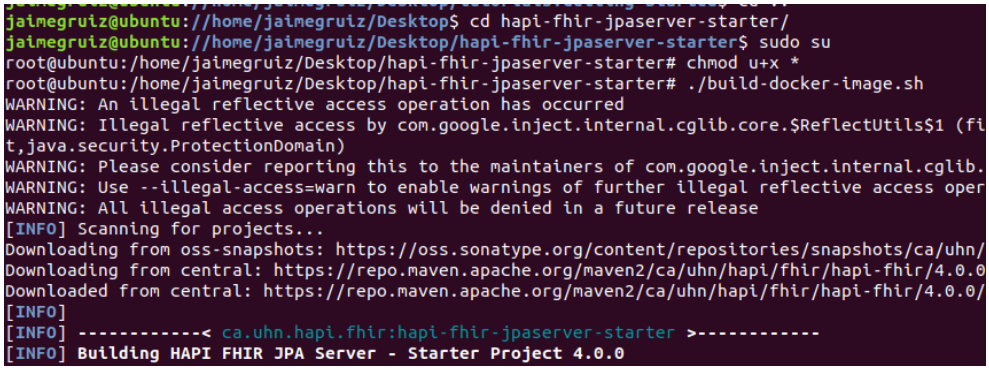

Figura 130. Ejecución "build-docker-image.sh"

En este caso Maven ha tardado unos 3 minutos y medio en descargar todas las dependencias necesarias y en montar el paquete war con la aplicación lista para ser ejecutada.

Figura 131. Resultado "build-docker-image.sh"

Tras terminar Maven, Docker empieza a ejecutar las instrucciones descritas previamente del archivo Dockerfile. Se puede observar en la captura como realiza los pasos de 1 al 4.

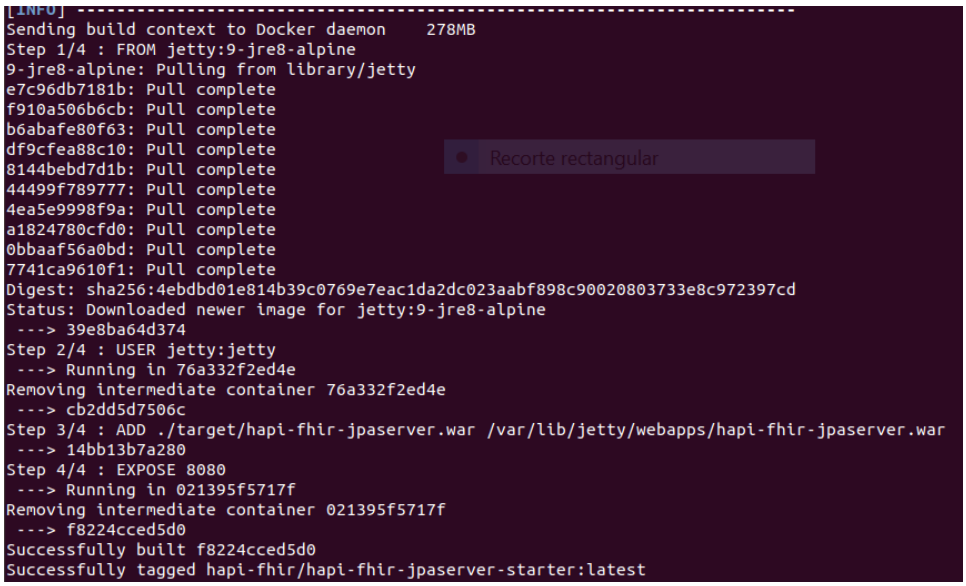

Figura 132. Ejecución de Dockerfile

Efectivamente, si se ejecuta el comando “Docker image ls” para listar las imágenes Docker cargadas en el equipo se puede ver la imagen recién creada “hapi-fhir/hapi-fhir-jpaserver-starter”.

```
root@ubuntu:/home/jaimegruiz/Desktop/hapi-fhir-jpaserver-starter# docker image ls
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
hapi-fhir/hapi-fhir-jpaserver-starter     latest              f8224cced5d0        22 minutes ago      223MB
fiware/orion                              latest              65743a72bfdd        6 days ago          280MB
mongo                                     3.6                 57c2f7e05108        3 weeks ago         434MB
jetty                                     9-jre8-alpine       39e8ba64d374        4 months ago        95.9MB
fiware/orion                              2.2.0               75c09d6a111f        6 months ago        271MB
hello-world                               latest              fce289e99eb9        8 months ago        1.84kB
```

Figura 133. Imágenes Docker

Mediante el siguiente comando se monta un contenedor Docker con la imagen recién creada.

```
root@ubuntu:/home/jaimegruiz/Desktop/hapi-fhir-jpaserver-starter# docker container run -it
--publish 8080:8080 hapi-fhir/hapi-fhir-jpaserver-starter
```

Figura 134. Lanzamiento de la imagen sobre un container

En la dirección http://localhost:8080/hapi-fhir-jpaserver/ encontraremos el servidor FHIR.

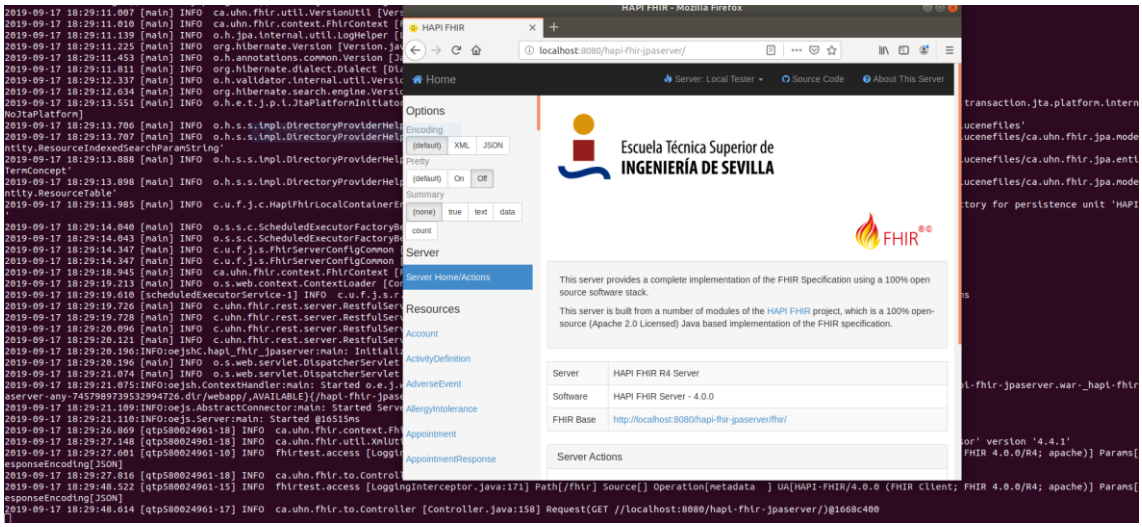

Figura 135. Ejecución y página de inicio del servidor FHIR

4.2 Orion Context Broker de FIWARE y MongoDB.

Se van a usar contenedores Docker para ejecutar tanto la base de datos MongoDB como el Context Broker. El único prerrequisito va a ser descargar el programa de la página oficial, donde se encuentran tutoriales para los diferentes sistemas operativos. En este caso se ha descargado siguiendo el tutorial de la página: https://docs.docker.com/install/linux/docker-ce/ubuntu/.

El proceso de instalación es muy sencillo, una vez instalado se puede comprobar la versión y ejecutar algún ejemplo para comprobar que todo está bien.

En primer lugar, hay que actualizar el índice de paquetes.

```
jaimegruiz@ubuntu:~$ sudo apt-get update
[sudo] password for jaimegruiz:
Hit:1 https://download.docker.com/linux/ubuntu bionic InRelease
Hit:2 http://ppa.launchpad.net/linuxuprising/java/ubuntu bionic InRelease
Get:3 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:4 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Get:5 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
```

Figura 136. Actualización índice de paquetes

En segundo lugar, hay que instalar los paquetes que permiten a la herramienta apt usar un repositorio sobre HTTPS

```
jaimegruiz@ubuntu:~$ sudo apt-get install \
>     apt-transport-https \
>     ca-certificates \
>     curl \
>     gnupg-agent \
>     software-properties-common
Reading package lists... Done
Building dependency tree
Reading state information... Done
ca-certificates is already the newest version (20180409).
curl is already the newest version (7.58.0-2ubuntu3.8).
software-properties-common is already the newest version (0.96.24.32.11).
apt-transport-https is already the newest version (1.6.12).
gnupg-agent is already the newest version (2.2.4-1ubuntu1.2).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
```

Figura 137. Paquetes para usar un repositorio sobre HTTPS

Se añade la clave GPG oficial de Docker y se comprueba que la huella corresponde con 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88.

```
jaimegruiz@ubuntu:~$  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-k
ey add -
OK
jaimegruiz@ubuntu:~$ sudo apt-key fingerprint 0EBFCD88
pub   rsa4096 2017-02-22 [SCEA]
      9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid           [ unknown] Docker Release (CE deb) <docker@docker.com>
sub   rsa4096 2017-02-22 [S]
```

Figura 138. Comprobación de la huella

Finalmente, se utiliza el siguiente comando para preparar Docker usando el repositorio marcado como “stable”. Con esto, se termina la preparación para usar el repositorio Docker.

```
jaimegruiz@ubuntu:~$ sudo add-apt-repository \
>    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
>    $(lsb_release -cs) \
>    stable"
Hit:1 https://download.docker.com/linux/ubuntu bionic InRelease
Get:2 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:3 http://ppa.launchpad.net/linuxuprising/java/ubuntu bionic InRelease
Hit:4 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Get:5 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:7 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [729 kB]
Get:8 http://us.archive.ubuntu.com/ubuntu bionic-updates/main i386 Packages [581 kB]
Get:9 http://us.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [1,003 kB
]
Get:10 http://us.archive.ubuntu.com/ubuntu bionic-updates/universe i386 Packages [979 kB]
Fetched 3,544 kB in 13s (269 kB/s)
Reading package lists... Done
```

Figura 139. Repositorio de Docker "stable"

Ahora se actualiza de nuevo el índice de paquetes apt y se instala la última versión de Docker.

```
jaimegruiz@ubuntu:~$ sudo apt-get update
Hit:1 https://download.docker.com/linux/ubuntu bionic InRelease
Hit:2 http://ppa.launchpad.net/linuxuprising/java/ubuntu bionic InRelease
Get:3 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:4 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Hit:5 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease
Get:6 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Fetched 163 kB in 5s (32.0 kB/s)
Reading package lists... Done
jaimegruiz@ubuntu:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  aufs-tools cgroupfs-mount pigz
The following NEW packages will be installed:
  aufs-tools cgroupfs-mount containerd.io docker-ce docker-ce-cli pigz
0 upgraded, 6 newly installed, 0 to remove and 3 not upgraded.
Need to get 88.0 MB of archives.
After this operation, 390 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Figura 140. Instalación Docker

Para verificar que se ha instalado correctamente, se descarga y se prueba la imagen “hello-world”. También se puede comprobar la versión instalada.

```
jaimegruiz@ubuntu:~$ docker --version
Docker version 19.03.2, build 6a30dfc
jaimegruiz@ubuntu:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figura 141. Versión y prueba de Docker

Al instalar Docker en Windows o Mac, por defecto también se instala la herramienta Docker Compose. Esta herramienta usa un archivo de configuración YAML para desplegar de forma rápida arquitecturas multi-contenedor. En Linux se instala de la siguiente forma.

En primer lugar, hay que ejecutar el siguiente comando para descargar la última versión estable de Docker Compose. Después se le aplican permisos de ejecución y se comprueba la instalación.

```
jaimegruiz@ubuntu:~$ sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/doc
ker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   617    0   617    0     0   1606      0 --:--:-- --:--:-- --:--:--  1602
100 15.4M  100 15.4M    0     0  4774k      0  0:00:03  0:00:03 --:--:-- 7363k
jaimegruiz@ubuntu:~$ ls
Desktop    Downloads  eclipse-workspace  Music     Postman  Templates
Documents  eclipse    examples.desktop   Pictures  Public   Videos
jaimegruiz@ubuntu:~$ sudo chmod +x /usr/local/bin/docker-compose
jaimegruiz@ubuntu:~$ docker-compose --version
docker-compose version 1.24.1, build 4667896b
jaimegruiz@ubuntu:~$
```

Figura 142. Descarga de Docker Compose

A continuación, se va a explicar cómo conseguir el Context Broker y la base de datos de dos formas, una usando solo Docker y la otra con Docker Compose. Esta es la arquitectura que queremos conseguir:

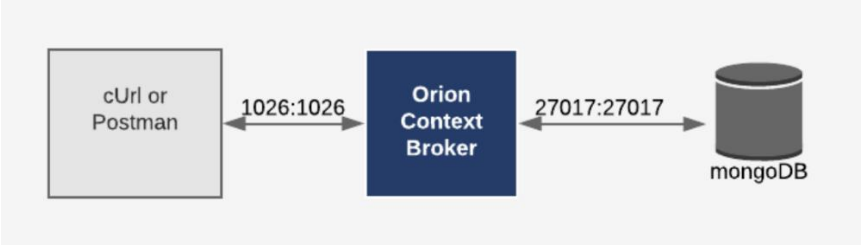

Figura 143. Arquitectura Orion + MongoDB

Hay que recordar que la base de datos MongoDB será accedida tanto por el Orion Context Broker como por la aplicación Java.

4.2.1 Primera Opción: Usando comandos de Docker

En primer lugar, hay que sacar las imágenes Docker de Orion y de MongoDB del Docker Hub [35]. Docker Hub es una librería enorme de imágenes de contenedores, además de una activa comunidad de desarrolladores. Proporciona una forma fácil de crear, controlar y compartir contenedores. Además de descargar las imágenes de los contenedores, creamos una red para conectarlos.

```
jaimegruiz@ubuntu:~$ sudo docker pull mongo:3.6
3.6: Pulling from library/mongo
Digest: sha256:4deb3044891f8e852dc59e013d304bacc9001f4ea5dcc4a81cc062758368cf09
Status: Image is up to date for mongo:3.6
docker.io/library/mongo:3.6
jaimegruiz@ubuntu:~$ sudo docker pull fiware/orion
Using default tag: latest
latest: Pulling from fiware/orion
Digest: sha256:b8d1870798d06e79b004c73de126f30d14fb08811e3687b46df28f6125f861ba
Status: Image is up to date for fiware/orion:latest
docker.io/fiware/orion:latest
jaimegruiz@ubuntu:~$ sudo docker network create fiware_default
Error response from daemon: network with name fiware_default already exists
```

Figura 144. Primera opción para lanzar Docker y MongoDB

Para empezar a ejecutar una base de datos MongoDB y conectarla a la red "fiware_default" que se acaba de crear se introduce el siguiente comando:

```
jaimegruiz@ubuntu:~$ sudo docker run -d --name=mongo-db --network=fiware_default --expose=27017
 mongo:3.6 --bind_ip_all --smallfiles
8ce0dc4a43ae26ec39b32e3dad545fc57aab3e4cb6e3cab61a0d52f2a4ea0917
```

Figura 145. Lanzamiento contenedor MongoDB

Para iniciar la imagen del Orion Context Broker y conectarla a la red se introduce el siguiente comando:

```
jaimegruiz@ubuntu:~$ sudo docker run -d --name fiware-orion -h orion --network=fiware_default
 -p 1026:1026 fiware/orion -dbhost mongo-db
6f7cd2fd536fb760f7bcdcd337caa32a27ca83d6f0072f94462e5ad182acddfb
```

Figura 146. Lanzamiento contenedor Orion

Con esto se tendría el escenario montado. Si se realiza una petición GET al Broker obtenemos la siguiente respuesta:

```
jaimegruiz@ubuntu:~$ curl --include \
>  --request GET \
> 'http://localhost:1026/version'
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Length: 345
Content-Type: application/json
Fiware-Correlator: 72489212-d972-11e9-88c3-0242ac120003
Date: Tue, 17 Sep 2019 17:41:57 GMT

{
"orion" : {
  "version" : "2.2.0-next",
  "uptime" : "0 d, 0 h, 1 m, 15 s",
  "git_hash" : "72390e5548b13d11c022904b322599ea5016336c",
  "compile_time" : "Wed Sep 11 12:51:28 UTC 2019",
  "compiled_by" : "root",
  "compiled_in" : "9a52085dcd0b",
  "release_date" : "Wed Sep 11 12:51:28 UTC 2019",
  "doc" : "https://fiware-orion.rtfd.io/"
}
}
```

Figura 147. Petición de prueba GET a Orion

Si se quisiera limpiar todo esto que se acaba de hacer y empezar de nuevo, lo haríamos con los siguientes comandos:

```
docker stop fiware-orion
docker rm fiware-orion
docker stop mongo-db
docker rm mongo-db
docker network rm fiware_default
```

Figura 148. Comando para parar y eliminar los contenedores

4.2.2 Segunda Opción: Usando la herramienta Docker Compose

En este caso se usa la herramienta de la línea de comandos Docker Compose. Es más rápido, pero hay que descargar primero los archivos de configuración de Github: https://github.com/FIWARE/tutorials.Getting-Started. Y después ejecutar el comando **"Docker-compose -p fiware up -d"**

```
jaimegruiz@ubuntu://home/jaimegruiz/Desktop$ git clone https://github.com/FIWARE/tutorials.Getting-Started
Cloning into 'tutorials.Getting-Started'...
remote: Enumerating objects: 56, done.
remote: Counting objects: 100% (56/56), done.
remote: Compressing objects: 100% (40/40), done.
remote: Total 475 (delta 30), reused 37 (delta 16), pack-reused 419
Receiving objects: 100% (475/475), 193.67 KiB | 833.00 KiB/s, done.
Resolving deltas: 100% (282/282), done.
jaimegruiz@ubuntu://home/jaimegruiz/Desktop$ cd tutorials.Getting-Started/
jaimegruiz@ubuntu://home/jaimegruiz/Desktop/tutorials.Getting-Started$ sudo docker-compose -p fiware up -d
Creating network "fiware_default" with the default driver
Creating db-mongo ... done
Creating fiware-orion ... done
```

Figura 149. Lanzamiento usando Docker Compose

Con esto se tendría el escenario montado. Si se realiza una petición GET al Broker se obtiene la misma respuesta que antes:

```
[]jaimegruiz@ubuntu://home/jaimegruiz/Desktop/tutorials.Getting-Started$ curl --include  --request
ET 'http://localhost:1026/version'
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Length: 348
Content-Type: application/json
Fiware-Correlator: da50c644-d973-11e9-8053-0242ac120103
Date: Tue, 17 Sep 2019 17:52:01 GMT

{
"orion" : {
  "version" : "2.2.0",
  "uptime" : "0 d, 0 h, 1 m, 5 s",
  "git_hash" : "5a46a70de9e0b809cce1a1b7295027eea0aa757f",
  "compile_time" : "Mon Feb 25 15:15:27 UTC 2019",
  "compiled_by" : "root",
  "compiled_in" : "37fdc92c3e97",
  "release_date" : "Mon Feb 25 15:15:27 UTC 2019",
  "doc" : "https://fiware-orion.rtfd.io/en/2.2.0/"
}
}
```

Figura 150. Petición de prueba GET a Orion

Si quisiéramos limpiar todo esto que se acaba de hacer y empezar de nuevo, se haría con el siguiente comando:

```
docker-compose -p fiware down
```

Figura 151. Comando Docker Compose para deshacer el lanzamiento

4.3 Jar de la aplicación y opciones de ejecución

Se va a crear un jar ejecutable de la aplicación Spring Boot con todas las dependencias necesarias para su ejecución independiente y autocontenida. Para lograr esto, hay que añadir el plugin "spring-boot-maven-plugin" al fichero "pom.xml" como se muestra en la siguiente imagen.

```
<build>
    <plugins>
     <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
            <mainClass>com.example.demo.Application</mainClass>
            <layout>JAR</layout>
        </configuration>
    </plugin>
```

Figura 152. "spring-boot-maven-plugin"

En la configuración se indica dónde se encuentra la clase que contiene el main de la aplicación, así como el formato al que se quiere exportar la aplicación.

Una vez hecho esto, hay que elegir la opción "Maven build…" dentro del menú Run.

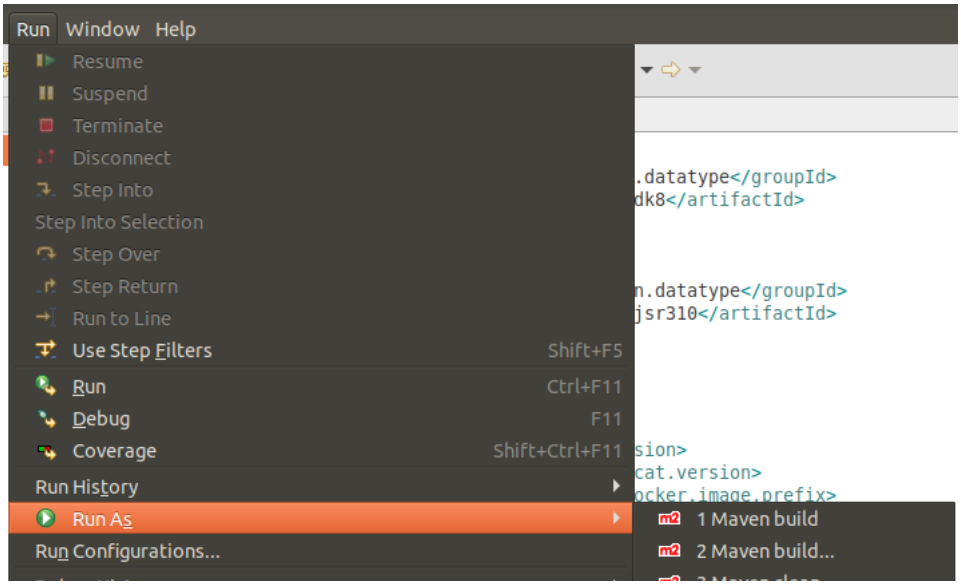

Figura 153. Menú Run Eclipse

Después, en la ventana de configuración hay que especificar el apartado Goals, este debe ser igual a “package”. Finalmente, ejecutamos la orden con Run. Entonces se observará en la consola de Eclipse como Maven se encarga en empaquetar el jar.

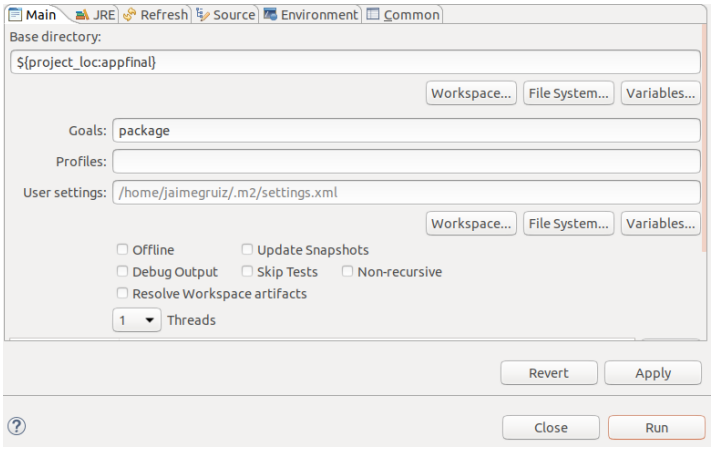

Figura 154. Ventana de configuración Maven build

```
Console
<terminated> APPFINAL (1) [Maven Build] /usr/lib/jvm/java-12-oracle/bin/java (Sep 18, 2019, 8:21:41 AM)
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.example:appfinal >------------------------
[INFO] Building appfinal 0.1.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]

[INFO] Downloading from : https://repo.maven.apache.org/maven2/com/google/guava/guava/19.0/guava-19.0.jar
[INFO] Layout: JAR
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  17.602 s
[INFO] Finished at: 2019-09-18T08:22:00-07:00
[INFO] ------------------------------------------------------------------------
```

Figura 155. Ejecución Maven Build

Se encontrará el jar empaquetado dentro de la carpeta target, localizada en la raíz del proyecto Maven.

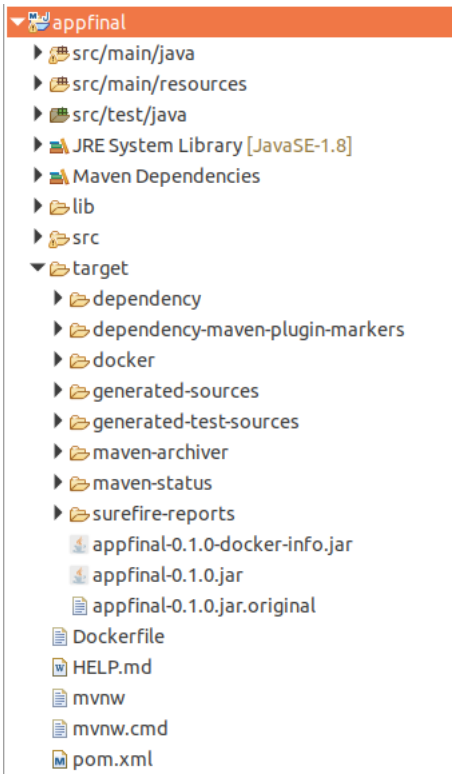

Figura 156. Explorador de proyectos de Eclipse

Se puede lanzar la aplicación con los parámetros por defecto usando el comando "java -jar appfinal-0.1.0.jar" si estamos situados en el directorio target.

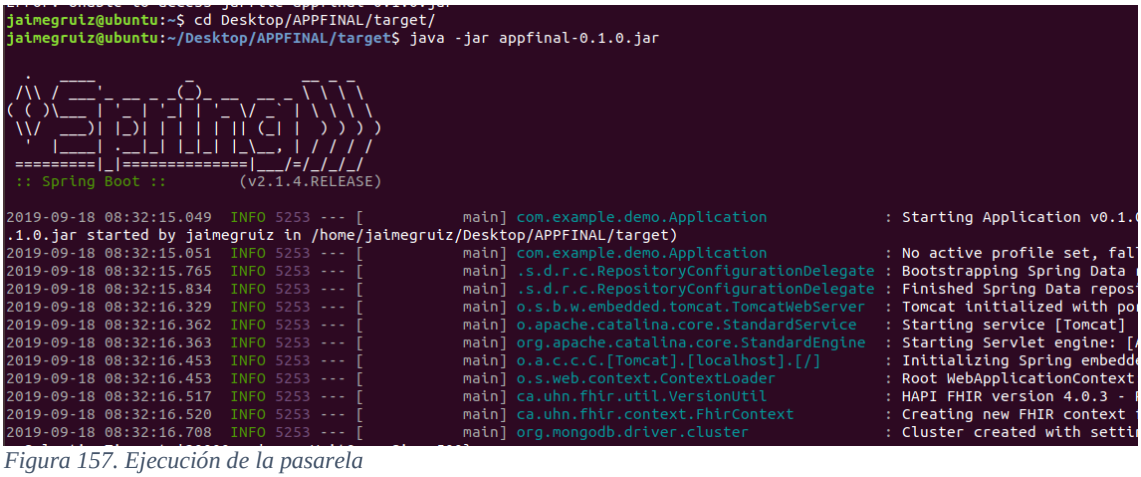

Figura 157. Ejecución de la pasarela

4.3.1 Parámetros configurables el ejecutar el jar

La aplicación arranca con unos valores por defecto ajustados para el entorno de pruebas, donde todos los servicios corren en la misma máquina. Se pueden configurar los siguientes parámetros:

- **server.port** : puerto del contenedor Tomcat donde se ejecuta la aplicación SpringBoot.
- **server.host** :dirección IP de la máquina donde se ejecuta la aplicación SprinBoot. Es necesario para saber dónde enviar las notificaciones HTTP POST de FIWARE.

- **fhir.address :** dirección completa, de la forma http:/address:port/baseURL. Se hago así ya que cada servidor FHIR implementa una URL base diferente.
- **fiware.host :** dirección IP del Context Broker de FIWARE.
- **fiware.port :** puerto del Context Broker de FIWARE.
- **spring.data.mongodb.port :** puerto de la base de datos MongoDB
- **spring.data.mongodb.host :** dirección IP de la base de datos MongoDB

Estos son los parámetros por defecto:

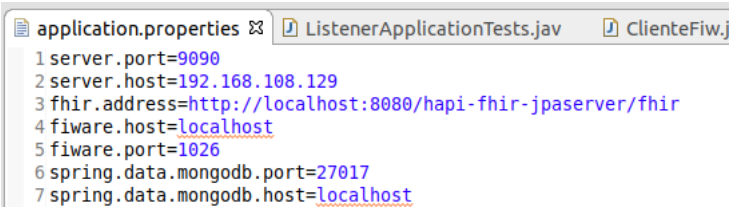

Figura 158. "application.properties"

Para pasar algún parámetro a la hora de ejecutar el jar, se hace de la siguiente forma:

Figura 159. Ejemplo de paso de párametros

Hay que tener en cuenta que los parámetros que no se especifiquen tomarán el valor por defecto arriba mostrado. Cuando se toman todos los valores por defecto, todos los servicios corren dentro de la misma máquina en localhost. Mostrado en un diagrama de despliegue quedaría así, las pruebas se han realizado usando este esquema:

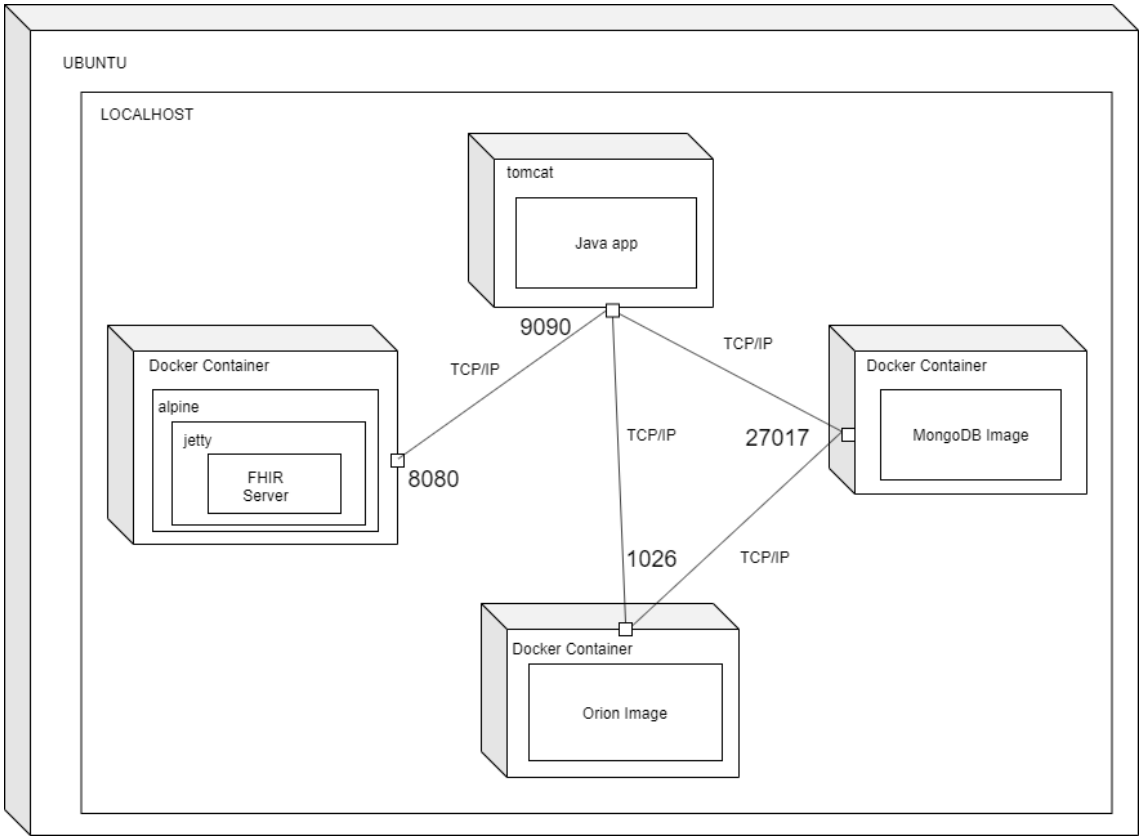

Figura 160. Diagrama de despliegue

4.4 Pruebas

Para estas pruebas se va a crear una nueva familia de dispositivos, concretamente una familia de glucómetros. Además, se van a crear tres dispositivos concretos sobre los que se realizarán cambios, a través de FIWARE, para observar cómo se actualizan los datos en FHIR.

Antes de empezar, hay que lanzar el servidor FHIR, y los contenedores de MongoDB y Orion. Se han creado una serie de scripts para facilitar el proceso. Finalmente, se lanza la aplicación.

Open ▾ runBrokerMongo.sh ~/Desktop Save

```
#!/bin/sh
cd //home/jaimegruiz/Desktop/tutorials.Getting-Started/
docker-compose -p fiware up -d
```

Figura 161. "runBrokerMongo.sh"

Open ▾ runFHIRServer.sh ~/Desktop Save

```
#!/bin/sh
docker container run -it --publish 8080:8080 hapi-fhir/
hapi-fhir-jpaserver-starter
```

Figura 162. "runFHIRServer.sh"

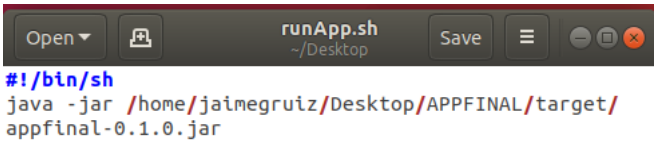

Figura 163. "runApp.sh"

Una vez que está todo montado, se va a usar la aplicación Postman para enviar los mensajes HTTP correspondientes. Los mensajes que tienen que ver con la gestión se envían a la interfaz REST de la pasarela. Los mensajes que simulan actualizaciones por parte de los sensores se envían al Orion Context Broker.

En primer lugar, hay que a crear la nueva familia de glucómetros en la pasarela. El proceso de creación de una nueva familia tiene 4 pasos en el siguiente orden:

1. Creación de la plantilla DeviceDefinition.
2. Creación de la plantilla Device.
3. Creación de la plantilla DeviceMetric.
4. Creación de la plantilla Observation.

En este caso, una plantilla no es más que un recurso que contiene información estática. Estos recursos se completarán con la información dinámica proporcionada por los dispositivos.

Como se ha comentado con anterioridad, gracias la correspondencia que se ha establecido entre los recursos FHIR y FIWARE, la aplicación es capaz de generar los recursos FIWARE Device y DeviceModel a partir de la información proporcionada por estas 4 plantillas. Por lo tanto, tras cumplir estos 4 pasos en orden, se deberían haber creado los recursos DeviceModel y Device en el Orion Context Broker. Esto es la configuración base con 1 solo dispositivo. Para añadir más dispositivos se enviarán más recursos o plantillas Device con distinto identificador, de igual forma se irán creando los recursos Device correspondientes en el Orion Context Broker.

4.4.1 Creación de la plantilla DeviceDefinition

El gestor envía un mensaje POST a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la pasarela de forma local y en el puerto 9090. La aplicación espera el recurso DeviceDefinition en la dirección "/devicedefinition".

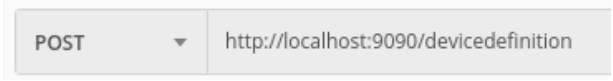

Figura 164. Petición creación DeviceDefinition

Se usan dos cabeceras HTTP, la "Accept" con valor "application/json" que indica que el cliente soporta una respuesta en formato JSON. Y la "Content-Type" con el mismo valor, indicando el tipo de contenido del cuerpo de la petición.

Figura 165. Cabecera HTTP creación DeviceDefinition

En el cuerpo del mensaje se encuentra el recurso DeficeDefinition en formato JSON siguiendo el estándar FHIR v4.0.0

```
{
  "resourceType": "DeviceDefinition",
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/DeviceDefinition",
      "value": "fiware_glucometer"
    }
  ],
  "manufacturerString" : "Accu-Chek",
  "deviceName": [
    {
      "name":"Glucose sensor 789",
      "type":"model-name"
    },
    {
      "name":"Glucometer",
      "type":"user-friendly-name"
    },
    {
        "name":"Accu-Chek",
        "type":"manufacturer-name"
    }

  ],
  "modelNumber" : "789",
  "type": {
    "coding": [
      {
        "system": "http://snomed.info/sct",
        "code": "701750003",
        "display": "Subcutaneous glucose sensor, device"
      }
    ]
  },
  "specialization" : [
    {
      "systemType" : "Bluetooth",
      "version" : "4.0"
    }
  ],
  "physicalCharacteristics" : {
      "shape" : "circular"
  }
}
```

Figura 166. Recurso DeviceDefinition en JSON

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK y un breve mensaje indicando el éxito de la petición.

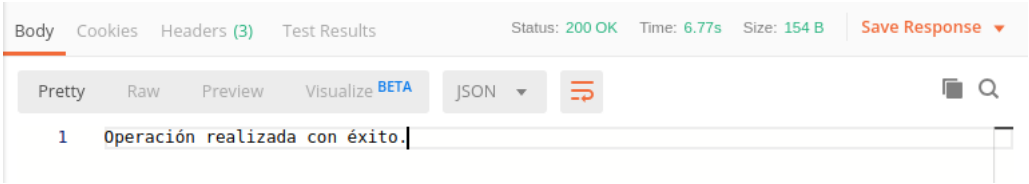

Figura 167. Respuesta exitosa creación DeviceDefinition

En el servidor FHIR, se verá que efectivamente el recurso ha sido creado.

Figura 168. Recurso DeviceDefinition en el servidor FHIR

Si el recurso DeviceDefinition que ha sido enviado en el cuerpo tiene errores de formato, el usuario recibe un mensaje 400 Bad Request de la pasarela y un breve mensaje de error.

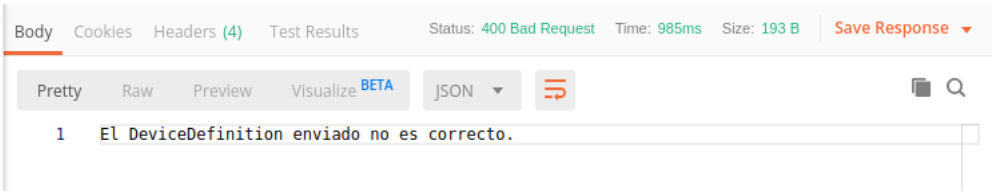

Figura 169. Respuesta en caso de error de formato

Además, si el usuario intenta crear un recurso DeviceDefinition ya existente en el sistema, recibe un mensaje de error 400 Bad Request de la pasarela y un mensaje como el siguiente.

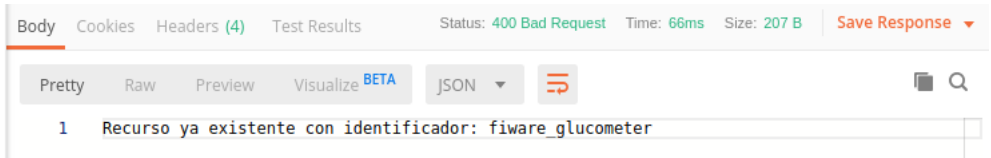

Figura 170. Respuesta en caso de DeviceDefinition ya existente

4.4.2 Creación de la plantilla Device

El gestor envía un mensaje POST a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la pasarela de forma local y en el puerto 9090. La aplicación espera el recurso Device en la dirección "/device".

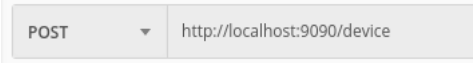

Figura 171. Petición creación Device

Se usan dos cabeceras HTTP, la “Accept” con valor “application/json” que indica que el cliente soporta una respuesta en formato JSON. Y la “Content-Type” con el mismo valor, indicando el tipo de contenido del cuerpo de la petición.

Figura 172. Cabera HTTP creación Device

En el cuerpo del mensaje se encuentra el recurso Device en formato JSON siguiendo el estándar FHIR v4.0.0


```
{
  "resourceType": "Device",
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/Device",
      "value": "fiware_glucometer_1"
    }
  ],
  "status": "active",
  "statusReason": [{"text":"online"}],
  "manufacturer": "Accu-Chek",
  "serialNumber":"12345678",
  "type": {
    "coding": [
      {
        "system": "http://snomed.info/sct",
        "code": "701750003",
        "display": "Subcutaneous glucose sensor, device"
      }
    ]
  },
  "deviceName": [
    {
      "name":"Glucose sensor 789",
      "type":"model-name"
    }
  ],
  "modelNumber" : "789",
  "version" :{
    "type": {
    "text":"Hardware"
    },
    "value": "2"
  },
  "manufactureDate": "2015-08-08",
  "url": "192.168.1.254"
}
```


Figura 173. Recurso Device en JSON

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK y un breve mensaje indicando el éxito de la petición.

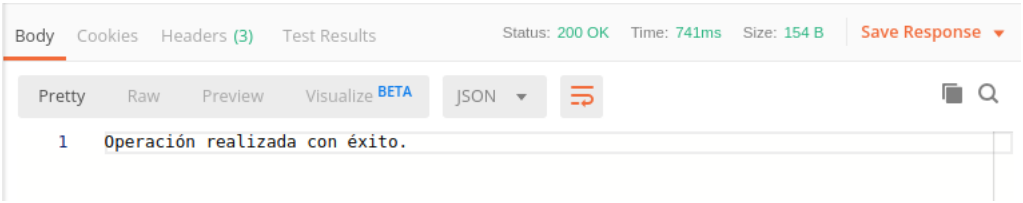

Figura 174. Respuesta exitosa creación Device

En el servidor FHIR, veremos que efectivamente los recursos Device y Location han sido creados.

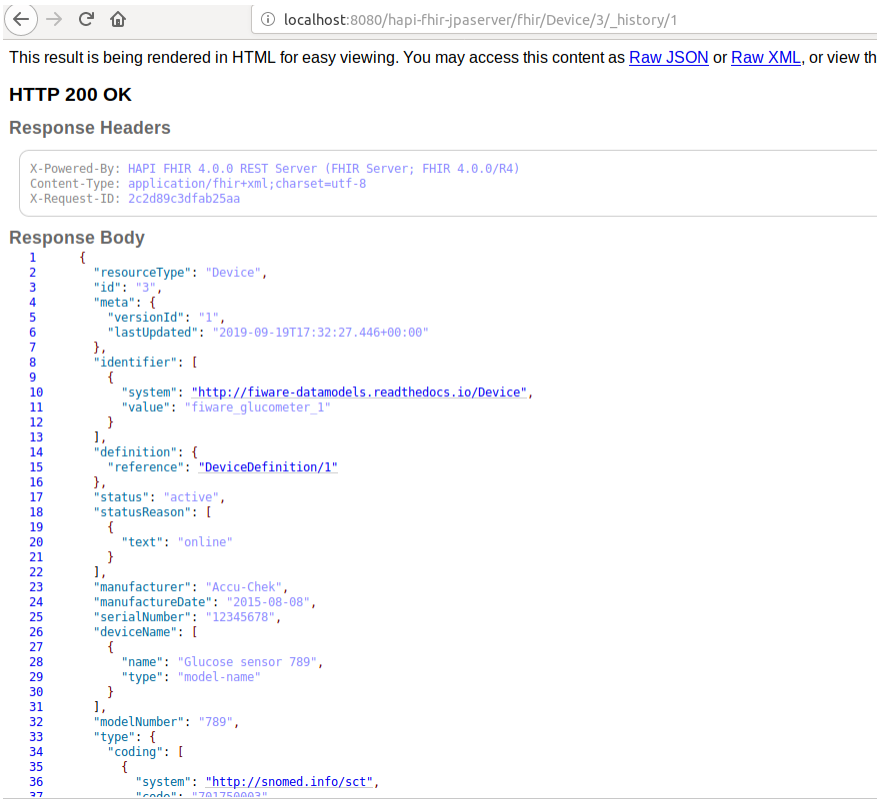

Figura 175. Recurso Device en el servidor FHIR

Figura 176. Recurso Location en el servidor FHIR

En el Orion Context Broker de FIWARE se comprueba que efectivamente la entidad ha sido creada.

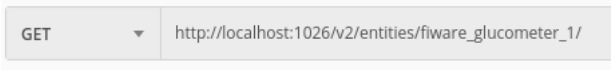

Figura 178. Petición GET a la entidad "fiware_glucometer_1"

Figura 177. Recurso Device FIWARE

Si el recurso Device enviado a la pasarela tiene errores de formato, el usuario recibe un mensaje 400 Bad Request y un breve mensaje de error.

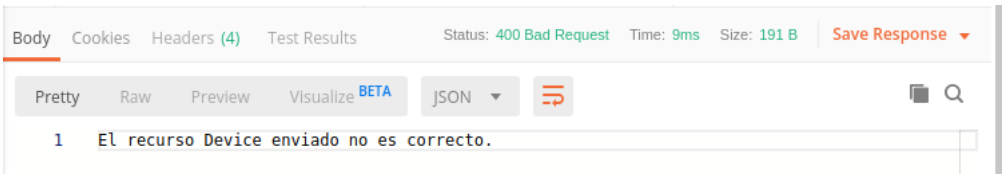

Figura 179. Respuesta en caso de error de formato del Device

Además, si el usuario intenta crear un recurso Device ya existente en el sistema recibe un mensaje de error 400 Bad Request y un mensaje como el siguiente.

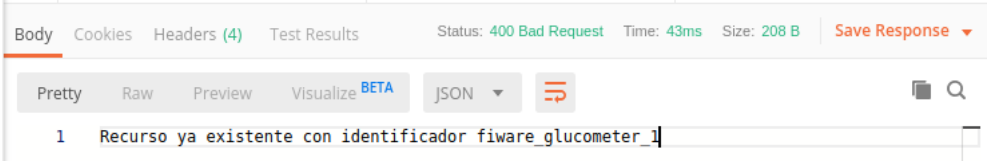

Figura 180. Respuesta en caso de Device ya existente

4.4.3 Creación de la plantilla DeviceMetric

El gestor envía un mensaje POST a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la pasarela de forma local y en el puerto 9090. La aplicación espera el recurso DeviceMetric en la dirección "/devicemetric".

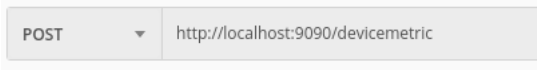

Figura 181. Petición creación DeviceMetric

Se usan dos cabeceras HTTP, la "Accept" con valor "application/json" que indica que el cliente soporta una respuesta en formato JSON. Y la "Content-Type" con el mismo valor, indicando el tipo de contenido del cuerpo de la petición.

Figura 182. Cabecera HTTP creación DeviceMetric

En el cuerpo del mensaje se encuentra el recurso DeviceMetric en formato JSON siguiendo el estándar FHIR v4.0.0 .


```
{
  "resourceType": "DeviceMetric",
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/DeviceMetric",
      "value": "fiware_glucometer_1_1"
    }
  ],
  "type": {
    "coding": [
      {
        "system": "urn:iso:std:iso:11073:10101",
        "code": "150456"
      }
    ]
  },
  "unit": {
    "coding": [
      {
        "system": "urn:iso:std:iso:11073:10101",
        "code": "28716",
        "display": "MDC_CONC_GLU_ART"
      }
    ]
  },
  "operationalStatus": "on",
  "category": "measurement",
  "measurementPeriod": {
    "repeat": {
      "frequency": 60,
      "periodUnit": "s"
    }
  },
  "calibration": [
    {
      "type": "unspecified",
      "state": "calibrated",
      "time": "2019-05-28T09:03:04-05:00"
    }
  ]
}
```


Figura 183. Recurso DeviceMetric en JSON

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK y un breve mensaje indicando el éxito de la petición.

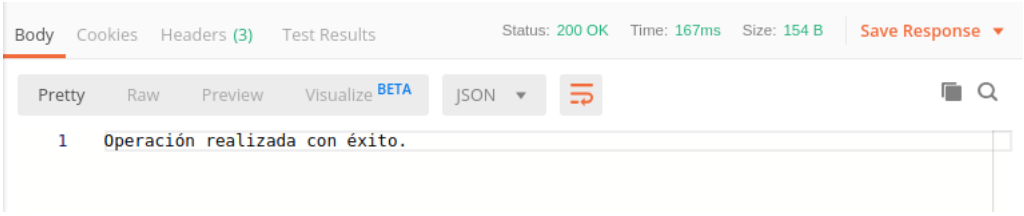

Figura 184. Respuesta exitosa creación DeviceMetric

En el servidor FHIR, se verá que efectivamente el recurso DeviceMetric ha sido creado.

Figura 185. Recurso DeviceMetric en el servidor FHIR

En FIWARE, se observa como finalmente se ha generado el recurso DeviceModel.

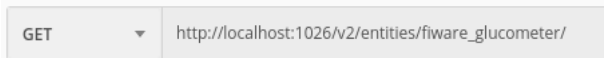

Figura 186. Petición GET a la entidad "fiware_glucometer"

```
{
    "id": "fiware_glucometer",
    "type": "DeviceModel",
    "brandName": {
        "type": "Text",
        "value": "Glucometer",
        "metadata": {}
    },
    "category": {
        "type": "StructuredValue",
        "value": [
            "Subcutaneous glucose sensor, device",
            "Bluetooth"
        ],
        "metadata": {}
    },
    "manufacturerName": {
        "type": "Text",
        "value": "Accu-Chek",
        "metadata": {}
    },
    "modelName": {
        "type": "Text",
        "value": "Glucose sensor 789",
        "metadata": {}
    },
    "supportedUnits": {
        "type": "StructuredValue",
        "value": [
            "MDC_CONC_GLU_ART"
        ],
        "metadata": {}
    }
}
```


Figura 187. Recurso DeviceModel

Si el recurso DeviceMetric enviado a la pasarela tiene errores de formato, el usuario recibe un mensaje 400 Bad Request y un breve mensaje de error.

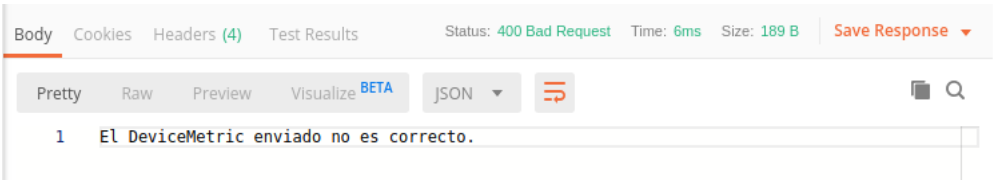

Figura 188. Respuesta en caso de error de formato del DeviceMetric

Además, si el usuario intenta crear un recurso Device ya existente en la pasarela, recibe un mensaje de error 400 Bad Request y un mensaje como el siguiente.

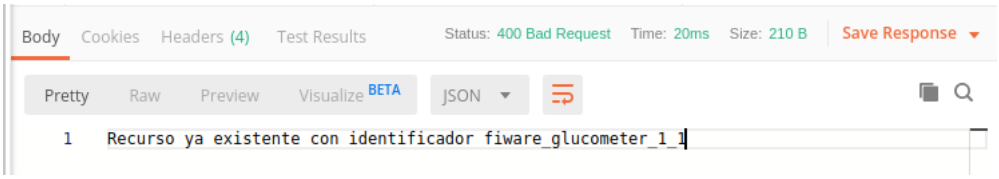

Figura 189. Respuesta en caso de DeviceMetric ya existente

4.4.4 Creación de la plantilla Observation

El gestor envía un mensaje POST a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la pasarela de forma local y en el puerto 9090. La aplicación espera el recurso Observation en la dirección "/observation".

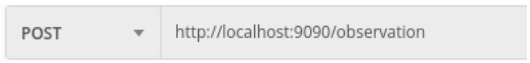

Figura 190. Petición creación Observation

Se usan dos cabeceras HTTP, la “Accept” con valor “application/json” que indica que el cliente soporta una respuesta en formato JSON. Y la “Content-Type” con el mismo valor, indicando el tipo de contenido del cuerpo de la petición.

Figura 191. Cabecera HTTP creación Observation

En el cuerpo del mensaje se encuentra el recurso DeficeDefinition en formato JSON siguiendo el estándar FHIR v4.0.0 .

```
{
  "resourceType": "Observation",
  "identifier": [
    {
      "system":"http://fiware-datamodels.readthedocs.io/Observation",
      "value": "fiware_glucometer_1_1"
    }
  ],
  "status" : "registered",
  "category": [
    {
      "coding": [
        {
          "system": "http://terminology.hl7.org/CodeSystem/observation-category",
          "code": "vital-signs"
        }
      ]
    }
  ],
  "code": {
    "coding": [
      {
        "system": "http://loinc.org",
        "code": "10449-7",
        "display": "Glucose [Mass/volume] in Serum or Plasma --1 hour post meal"
      }
    ],
    "text": "Levels of glucose"
  }
}
```

Figura 192. Recurso Observation en JSON

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con código 200 OK y un breve mensaje indicando el éxito de la petición.

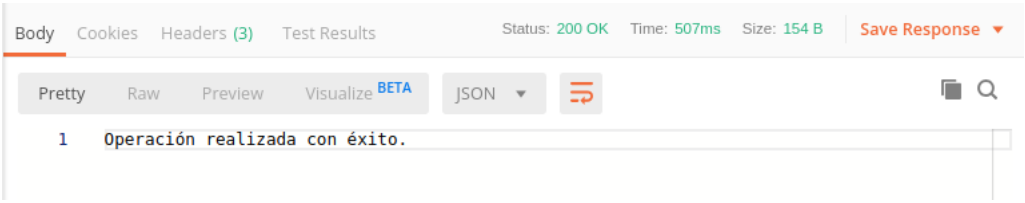

Figura 193. Respuesta exitosa creación Observation

Hasta este momento no se ha generado ningún recurso en el servidor FHIR, hasta que el dispositivo no haga la primera medición no se creará el primer recurso Observation.

En el Orion Context Broker FIWARE se puede comprobar que se han creado las 5 suscripciones clave para que funcione nuestro escenario.

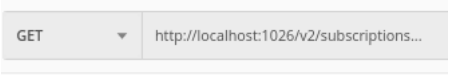

Figura 194. Petición GET a Orion de todas las suscripciones


```
{
    "id": "5d83bbb0996a4c7cc9104c1c",
    "description": "Suscripción a cambios en el valor del sensor",
    "expires": "2022-01-01T14:00:00.00Z",
    "status": "active",
    "subject": {
        "entities": [
            {
                "idPattern": "^fiware_glucometer",
                "type": "Device"
            }
        ],
        "condition": {
            "attrs": [
                "value"
            ]
        }
    },
    "notification": {
        "attrs": [
            "id",
            "value"
        ],
        "attrsFormat": "keyValues",
        "http": {
            "url": "http://192.168.108.129:9090/value"
        }
    },
```


Figura 196. Suscripción cambios valor medido


```
{
    "id": "5d83bbb0996a4c7cc9104c1d",
    "description": "Suscripción a cambios en el valor de la dirección IP",
    "expires": "2022-01-01T14:00:00.00Z",
    "status": "active",
    "subject": {
        "entities": [
            {
                "idPattern": "^fiware_glucometer",
                "type": "Device"
            }
        ],
        "condition": {
            "attrs": [
                "ipAddress"
            ]
        }
    },
    "notification": {
        "attrs": [
            "id",
            "ipAddress"
        ],
        "attrsFormat": "keyValues",
        "http": {
            "url": "http://192.168.108.129:9090/ipaddress"
        }
    },
    "throttling": 1
},
```


Figura 195. Suscripción cambios IP


```
{
    "id": "5d83bbb0996a4c7cc9104c1e",
    "description": "Suscripción a cambios en la localización del sensor",
    "expires": "2022-01-01T14:00:00.00Z",
    "status": "active",
    "subject": {
        "entities": [
            {
                "idPattern": "^fiware_glucometer",
                "type": "Device"
            }
        ],
        "condition": {
            "attrs": [
                "location"
            ]
        }
    },
    "notification": {
        "attrs": [
            "id",
            "location"
        ],
        "attrsFormat": "keyValues",
        "http": {
            "url": "http://192.168.108.129:9090/location"
        }
    },
    "throttling": 1
},
```


Figura 198. Suscripción a cambios en la localización


```
{
    "id": "5d83bbb0996a4c7cc9104c1f",
    "description": "Suscripción a cambios en la fecha de calibración del sensor",
    "expires": "2022-01-01T14:00:00.00Z",
    "status": "active",
    "subject": {
        "entities": [
            {
                "idPattern": "^fiware_glucometer",
                "type": "Device"
            }
        ],
        "condition": {
            "attrs": [
                "dateLastCalibration"
            ]
        }
    },
    "notification": {
        "attrs": [
            "id",
            "dateLastCalibration"
        ],
        "attrsFormat": "keyValues",
        "http": {
            "url": "http://192.168.108.129:9090/calibration"
        }
    },
    "throttling": 1
},
{
```


Figura 197. Suscripción a cambios en la fecha de calibración

```
{
    "id": "5d83bbb0996a4c7cc9104c20",
    "description": "Suscripción a cambios en el estado del sensor",
    "expires": "2022-01-01T14:00:00.00Z",
    "status": "active",
    "subject": {
        "entities": [
            {
                "idPattern": "^fiware_glucometer",
                "type": "Device"
            }
        ],
        "condition": {
            "attrs": [
                "deviceState"
            ]
        }
    },
    "notification": {
        "attrs": [
            "id",
            "deviceState"
        ],
        "attrsFormat": "keyValues",
        "http": {
            "url": "http://192.168.108.129:9090/state"
        }
    },
    "throttling": 1
}
```


Figura 199. Suscripción a cambios del estado

Si el recurso Observation enviado a la pasarela tiene errores de formato, el usuario recibe un mensaje 400 Bad Request y un breve mensaje de error.

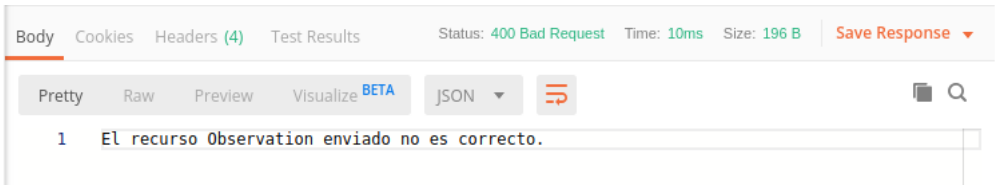

Figura 200. Respuesta en caso de error de formato del Observation

Además, si el usuario intenta crear un recurso Observation ya existente en la pasarela, recibe un mensaje de error 400 Bad Request y un mensaje como el siguiente.

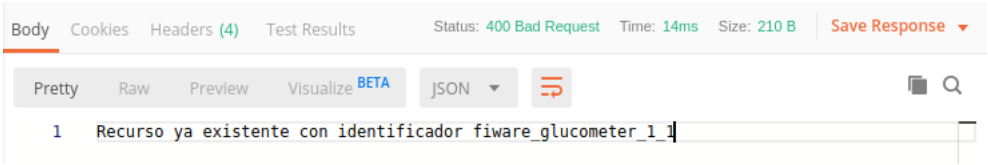

Figura 201. Respuesta en caso de Observation ya existente

4.4.5 Creación de recursos Device adicionales

El gestor envía un mensaje POST a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la pasarela de forma local y en el puerto 9090. La aplicación espera el recurso Device en la dirección "/deviceonly".

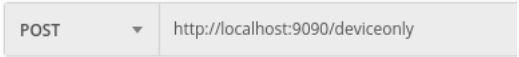

Figura 202. Petición creación Device adicional

Se usan dos cabeceras HTTP, la "Accept" con valor "application/json" que indica que el cliente soporta una respuesta en formato JSON. Y la "Content-Type" con el mismo valor, indicando el tipo de contenido del cuerpo de la petición.

Figura 203. Cabecera HTTP creación Device adicional

En el cuerpo del mensaje se encuentra el recurso Device en formato JSON siguiendo el estándar FHIR v4.0.0. En este caso se mandan dos mensajes a la pasarela variando su cuerpo, para añadir los dos dispositivos adicionales al sistema.

```
{
  "resourceType": "Device",
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/Device",
      "value": "fiware_glucometer_2"
    }
  ],
  "status": "active",
  "statusReason": [{"text":"online"}],
  "manufacturer": "Accu-Chek",
  "serialNumber":"12345679",
  "type": {
    "coding": [
      {
        "system": "http://snomed.info/sct",
        "code": "701750003",
        "display": "Subcutaneous glucose sensor, device"
      }
    ]
  },
  "deviceName": [
    {
      "name":"Glucose sensor 789",
      "type":"model-name"
    }
  ],
  "modelNumber" : "789",
  "version" :{
    "type": {
      "text":"Hardware"
    },
    "value": "3"
  },
  "manufactureDate": "2019-07-08",
  "url": "192.168.1.253"
}
```

```
{
  "resourceType": "Device",
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/Device",
      "value": "fiware_glucometer_3"
    }
  ],
  "status": "active",
  "statusReason": [{"text":"online"}],
  "manufacturer": "Accu-Chek",
  "serialNumber":"12345656",
  "type": {
    "coding": [
      {
        "system": "http://snomed.info/sct",
        "code": "701750003",
        "display": "Subcutaneous glucose sensor, device"
      }
    ]
  },
  "deviceName": [
    {
      "name":"Glucose sensor 789",
      "type":"model-name"
    }
  ],
  "modelNumber" : "789",
  "version" :{
    "type": {
      "text":"Hardware"
    },
    "value": "3"
  },
  "manufactureDate": "2018-07-08",
  "url": "192.168.1.250"
}
```

Figura 204. Recursos Device

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK y un breve mensaje indicando el éxito de la petición.

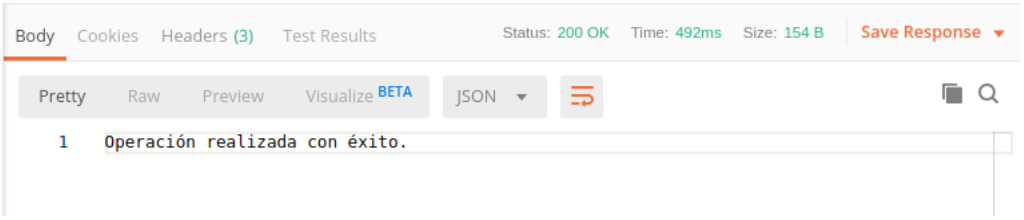

Figura 205. Respuesta en caso de éxito

En el servidor FHIR, se puede observar que se han creado dos recursos Device nuevos.

HTTP 200 OK

Response Headers

```
X-Powered-By: HAPI FHIR 4.0.0 REST Server (FHIR Server; FHIR 4.0.0/R4)
Content-Type: application/fhir+xml;charset=utf-8
X-Request-ID: 2bd1457d9f2e3472
```

Response Body


```
{
  "resourceType": "Device",
  "id": "10",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2019-09-19T21:55:13.175+00:00"
  },
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/Device",
      "value": "fiware_glucometer_2"
    }
  ],
  "definition": {
    "reference": "DeviceDefinition/1"
  },
  "status": "active",
  "statusReason": [
    {
      "text": "online"
    }
  ],
  "manufacturer": "Accu-Chek",
  "manufactureDate": "2019-07-08",
  "serialNumber": "12345679",
  "deviceName": [
    {
      "name": "Glucose sensor 789",
      "type": "model-name"
    }
  ],
  "modelNumber": "789",
  "type": {
    "coding": [
      {
        "system": "http://snomed.info/sct",
        "code": "701750003",
        "display": "Subcutaneous glucose sensor, device"
```


HTTP 200 OK

Response Headers

```
X-Powered-By: HAPI FHIR 4.0.0 REST Server (FHIR Server; FHIR 4.0.0/R4)
Content-Type: application/fhir+xml;charset=utf-8
X-Request-ID: 882726249694f2e0
```

Response Body


```
{
  "resourceType": "Device",
  "id": "6",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2019-09-19T21:55:00.085+00:00"
  },
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/Device",
      "value": "fiware_glucometer_3"
    }
  ],
  "definition": {
    "reference": "DeviceDefinition/1"
  },
  "status": "active",
  "statusReason": [
    {
      "text": "online"
    }
  ],
  "manufacturer": "Accu-Chek",
  "manufactureDate": "2018-07-08",
  "serialNumber": "12345656",
  "deviceName": [
    {
      "name": "Glucose sensor 789",
      "type": "model-name"
    }
  ],
  "modelNumber": "789",
  "type": {
    "coding": [
      {
        "system": "http://snomed.info/sct",
        "code": "701750003",
        "display": "Subcutaneous glucose sensor, device"
```


Figura 206. Recursos Device en el servidor FHIR

También se puede comprobar que el Orion Context Broker se han creado dos recursos Device.

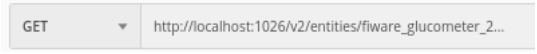

Figura 207. Petición GET a Orion del recurso "fiware_glucometer_2"

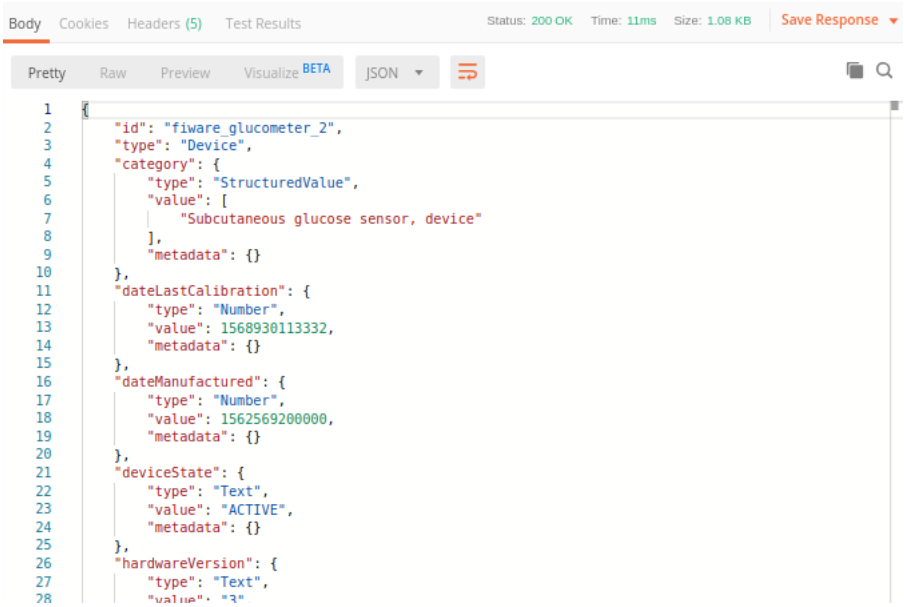

Figura 208. Recurso Device "fiware_glucometer_2"

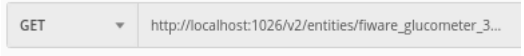

Figura 209. Petición GET a Orion del recurso "fiware_glucometer_3"

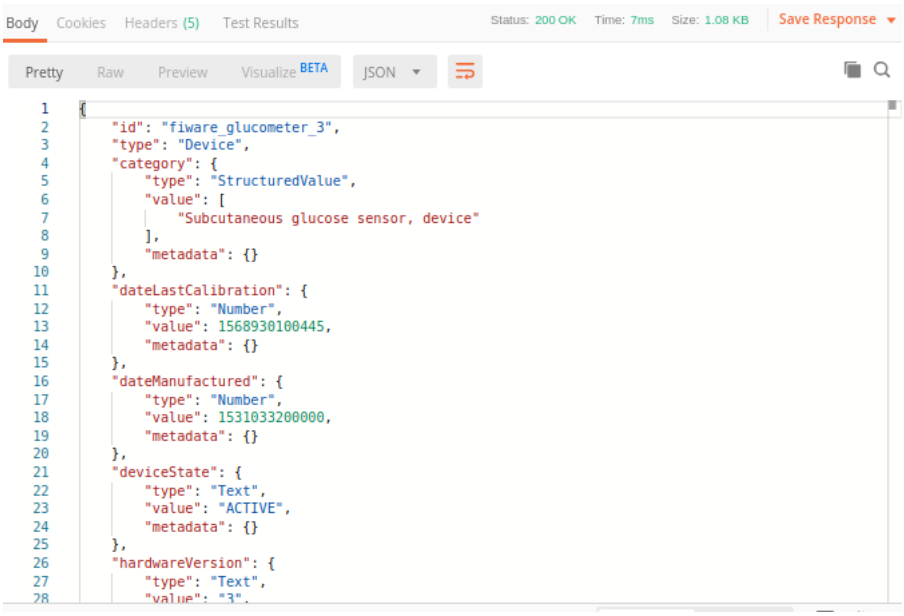

Figura 210. Recurso Device "fiware_glucometer_3"

Si el recurso Device enviado a la pasarela tiene errores de formato, el usuario recibe un mensaje 400 Bad Request y un breve mensaje de error.

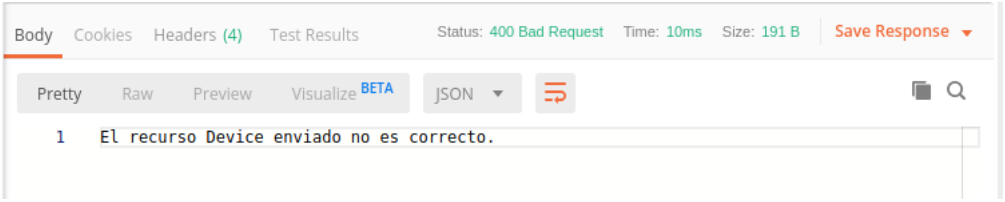

Figura 211. Respuesta en caso de error de formato del Device

Además, si el usuario intenta crear un recurso ya existente en el sistema recibe un mensaje de error 400 Bad Request y un mensaje como el siguiente.

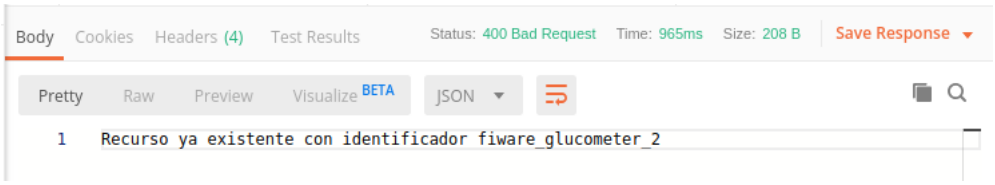

Figura 212. Respuesta en caso de Device ya existente

4.4.6 Actualización del valor medido por un dispositivo

Se envía un mensaje PUT al Orion Context Broker, así se simula que un dispositivo concreto ha notificado a Orion de un nuevo valor medido. El mensaje tiene las siguientes características.

Se usa la URL por defecto, ya que se ejecuta el Context Broker de forma local y en el puerto 1026. El recurso es accesible en la siguiente dirección según el estándar NGSIv2. Se quiere actualizar el valor del atributo value dentro de la entidad con identificador "fiware_glucometer_1".

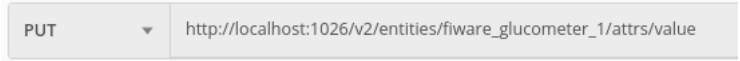

Figura 213. Actualización de "value" de "fiware_glucometer_1"

La única cabecera HTTP utilizada es la de “Content-Type”, indicando el formato JSON del cuerpo del mensaje.

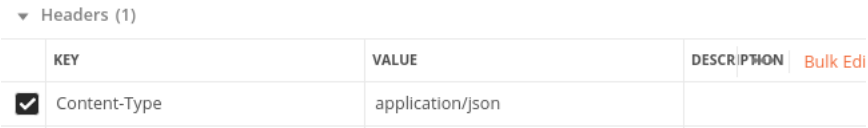

Figura 214. Cabecera HTTP actualización "value"

En el cuerpo del mensaje se encuentra el nuevo valor que mide el dispositivo.

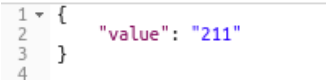

Figura 215. Nuevo "value"

En el servidor FHIR se ve cómo se ha creado el recurso Observation.

Figura 216. Recurso Observation en el servidor FHIR

4.4.7 Actualización de la localización de un dispositivo

Se envía un mensaje PUT al Orion Context Broker, así se simula que un dispositivo concreto ha notificado a Orion de una nueva posición. El mensaje tiene las siguientes características.

Se usa la URL por defecto, ya que se ejecuta el ContextBroker de forma local y en el puerto 1026. El recurso es accesible en la siguiente dirección según el estándar NGSIv2. Se quiere actualizar el valor del atributo location dentro de la entidad con identificador “fiware_glucometer_1”.

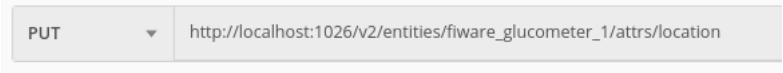

Figura 217. Actualización de "location" de "fiware_glucometer_1"

La única cabecera HTTP utilizada es la de “Content-Type”, indicando el formato JSON del cuerpo del mensaje.

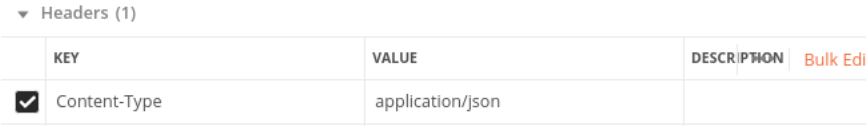

Figura 218. Cabecera HTTP actualización "location"

En el cuerpo del mensaje se encuentra la nueva posición del dispositivo.

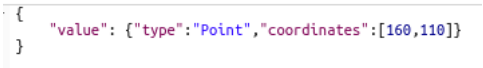

Figura 219. Nuevo "location"

En el servidor FHIR se verá que se ha actualizado el recurso Location.

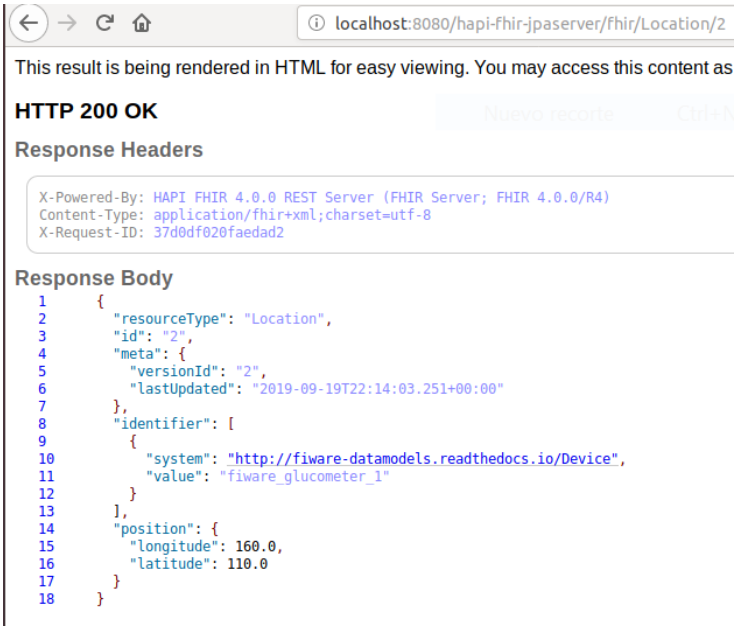

Figura 220. Recurso Location en el servidor FHIR

4.4.8 Actualización del estado de un dispositivo

Se envía un mensaje PUT al Orion Context Broker, así se simula que un dispositivo concreto ha notificado a Orion de su nuevo estado. El mensaje tiene las siguientes características.

Se usa la URL por defecto, ya que se ejecuta el ContextBroker de forma local y en el puerto 1026. El recurso es accesible en la siguiente dirección según el estándar NGSIv2. Se quiere actualizar el valor del atributo deviceState dentro de la entidad con identificador “fiware_glucometer_1”.

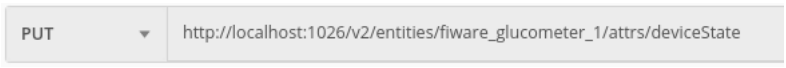

Figura 221. Actualización de "deviceState" de "fiware_glucometer_1"

La única cabecera HTTP utilizada es la de “Content-Type”, indicando el formato JSON del cuerpo del mensaje.

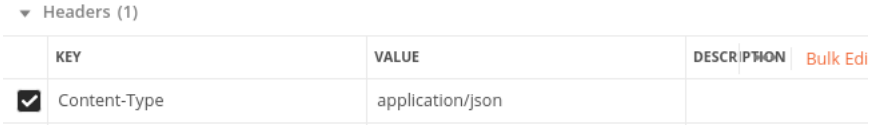

Figura 222. Cabecera HTTP actualización "deviceState"

En el cuerpo del mensaje se encuentra el nuevo valor que indica el estado del dispositivo.

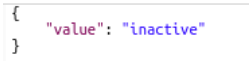

Figura 223. Nuevo "deviceState"

En el servidor FHIR se verá que se ha actualizado el recurso Device.

Figura 224. Recurso Device actualizado

4.4.9 Actualización de la fecha de calibración de un dispositivo

Se envía un mensaje PUT al Orion Context Broker, así se simula que un dispositivo concreto ha notificado a Orion de una nueva fecha de calibración. El mensaje tiene las siguientes características.

Se usa la URL por defecto, ya que se ejecuta el ContextBroker de forma local y en el puerto 1026. El recurso es accesible en la siguiente dirección según el estándar NGSIv2. Se quiere actualizar el valor del atributo dateLastCalibration dentro de la entidad con identificador “fiware_glucometer_1”.

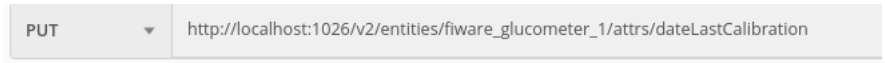

Figura 225. Actualización de "dateLastCalibration" de "fiware_glucometer_1"

La única cabecera HTTP utilizada es la de “Content-Type”, indicando el formato JSON del cuerpo del mensaje.

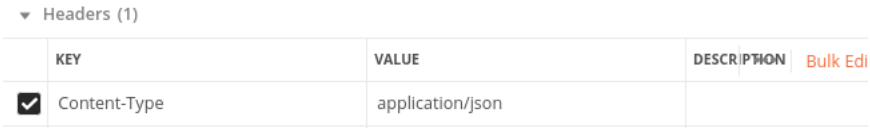

Figura 226. Cabecera HTTP actualización "dateLastCalibration"

En el cuerpo del mensaje se encuentra la nueva fecha de calibración del dispositivo.

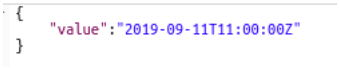

Figura 227. Nueva fecha de calibración

En el servidor FHIR, se verá que se ha actualizado el recurso DeviceMetric.

Figura 228. Recurso DeviceMetric actualizado en el servidor FHIR

4.4.10 Actualización de la dirección IP de un dispositivo

Se envía un mensaje PUT al Orion Context Broker, así se simula que un dispositivo concreto ha notificado a Orion de una nueva dirección IP. El mensaje tiene las siguientes características.

Se usa la URL por defecto, ya que se ejecuta el ContextBroker de forma local y en el puerto 1026. El recurso es accesible en la siguiente dirección según el estándar NGSIv2. Se quiere actualizar el valor del atributo ipAddress dentro de la entidad con identificador “fiware_glucometer_1”.

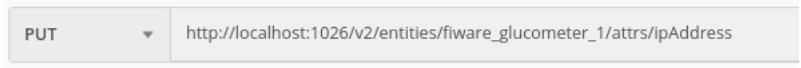

Figura 229. Actualización de "ipAddress" de "fiware_glucometer_1"

La única cabecera HTTP utilizada es la de “Content-Type”, indicando el formato JSON del cuerpo del mensaje.

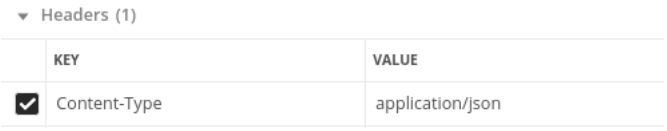

Figura 230. Cabecera HTTP actualización "ipAddress"

En el cuerpo del mensaje se encuentra la nueva dirección IP del dispositivo.

```
{
    "value": "192.168.14.23"
}
```

Figura 231. Nuevo "ipAddress"

En el servidor FHIR efectivamente el recurso Device ha sido modificado.

Figura 232. Recurso Device modificado en el servidor FHIR

4.4.11 Recuperación de un recurso Device previamente creado

El gestor envía un mensaje GET a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la pasarela de forma local y en el puerto 9090. La aplicación espera la petición en la dirección “/device/{identificador del recurso Device}”.

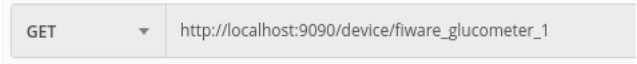

Figura 233. Petición recuperación Device

La única cabecera HTTP utilizada es la "Accept" con valor "application/json", que indica que el cliente soporta una respuesta en formato JSON.

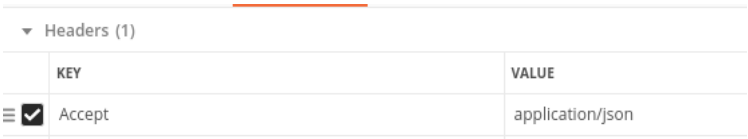

Figura 234. Cabecera HTTP recuperación Device

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK y con el recurso Device en el cuerpo de la respuesta codificado en JSON.

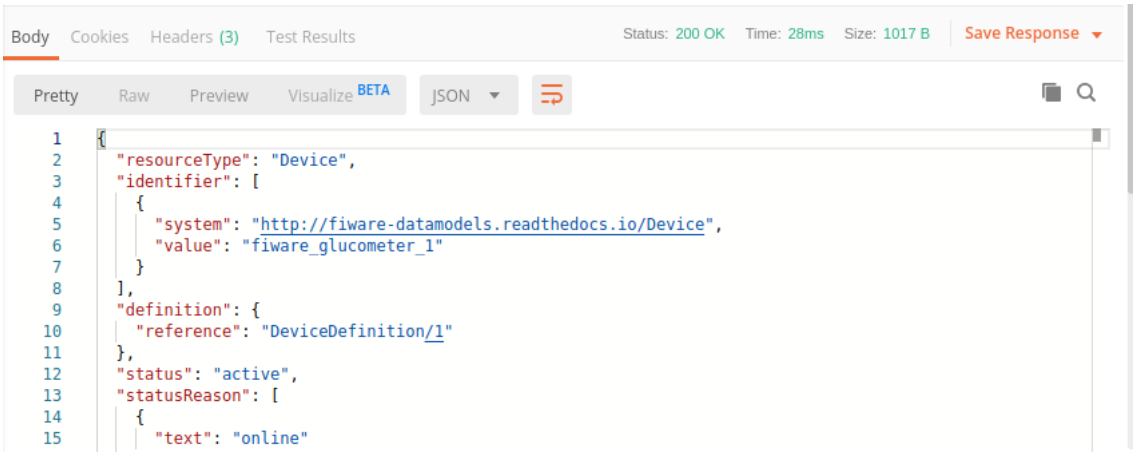

Figura 235. Fragmento del recurso Device recuperado

Si el usuario solicita un recurso Device con un identificador que no exista en la pasarela, el usuario recibe un mensaje 404 Not Found y un breve mensaje de error.

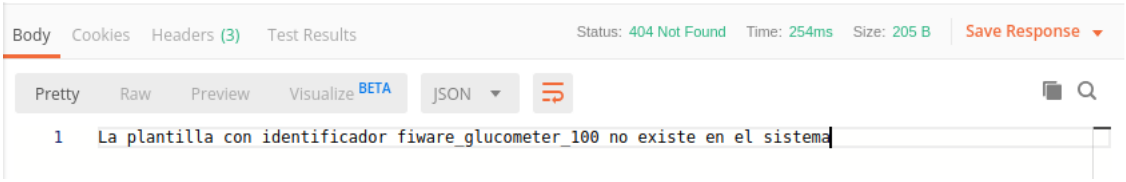

Figura 236. Respuesta en caso de que el Device solicitado no exista

4.4.12 Recuperación de un recurso DeviceDefinition previamente creado

Elgestor envía un mensaje GET a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la aplicación Spring de forma local y en el puerto 9090. La aplicación espera la petición en la dirección "/devicedefinition/{identificador de la familia de dispositivos}".

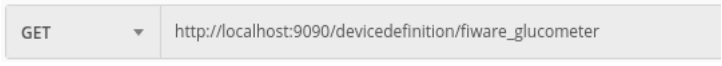

Figura 237. Petición recuperación DeviceDefinition

La única cabecera HTTP utilizada es la “Accept” con valor “application/json”, que indica que el cliente soporta una respuesta en formato JSON.

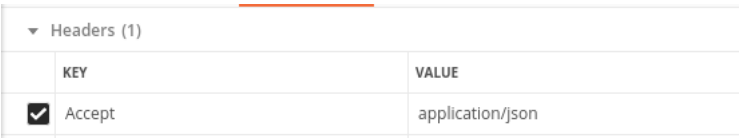

Figura 238. Cabera HTTP recuperación DeviceDefinition

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK y con el recurso DeviceDefinition en el cuerpo de la respuesta codificado en JSON.

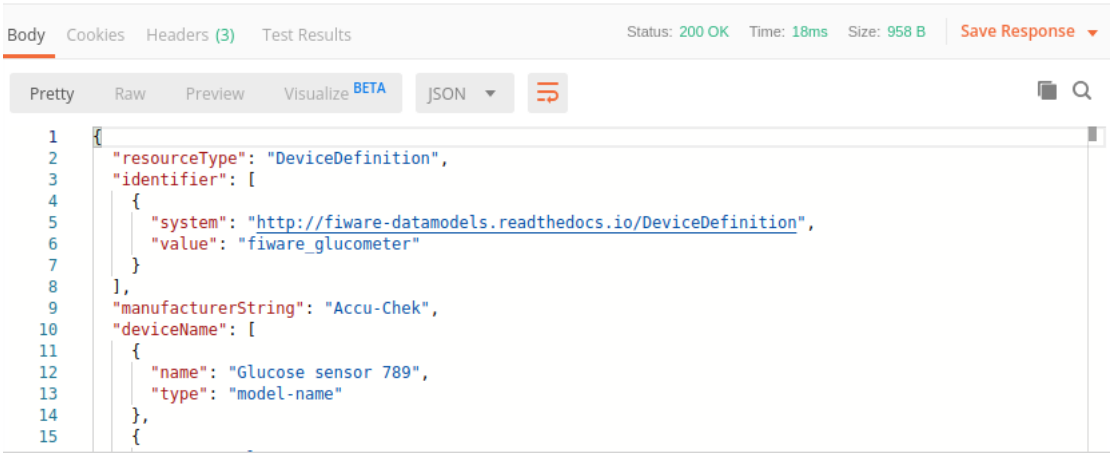

Figura 239. Fragmento del DeviceDefinition recuperado

Si el usuario solicita un recurso con un identificador que no exista en la pasarela, el usuario recibe un mensaje 404 Not Found y un breve mensaje de error.

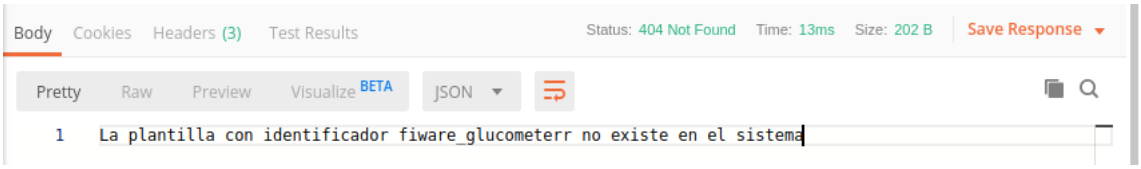

Figura 240. Respuesta en caso de que el DeviceDefinition solicitado no exista

4.4.13 Recuperación de un recurso DeviceMetric previamente creado

El gestor envía un mensaje GET a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la aplicación Spring de forma local y en el puerto 9090. La aplicación espera la petición en la dirección “/devicemetric/{identificador de la familia de dispositivos}”.

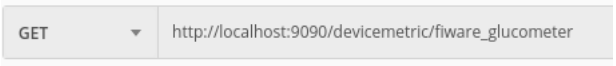

Figura 241. Petición recuperación DeviceMetric

La única cabecera HTTP utilizada es la “Accept” con valor “application/json”, que indica que el cliente soporta una respuesta en formato JSON.

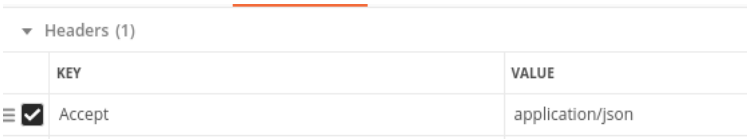

Figura 242. Cabecera HTTP recuperación DeviceMetric

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK y con el recurso DeviceMetric en el cuerpo de la respuesta codificado en JSON.

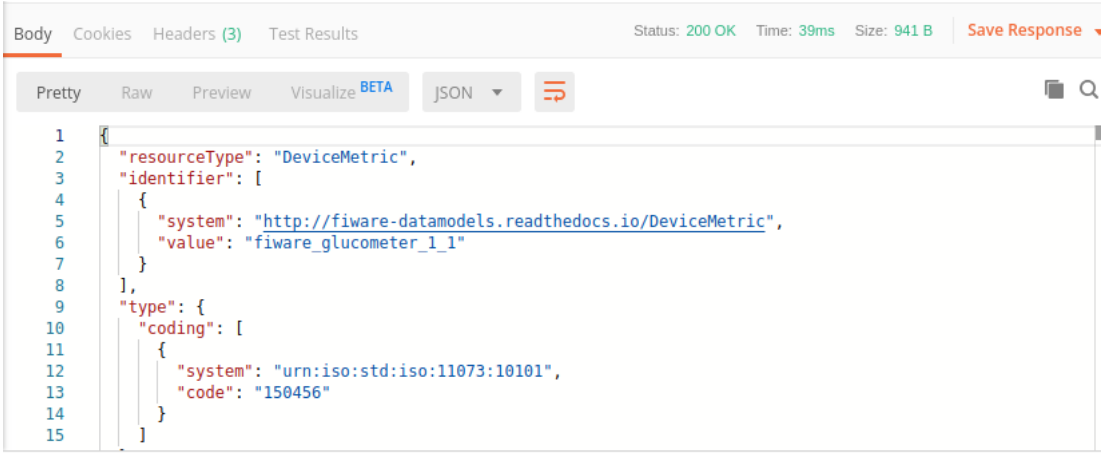

Figura 243. Fragmento del recurso DeviceMetric recuperado

Si el usuario solicita un recurso con un identificador que no exista en la pasarela, el usuario recibe un mensaje 404 Not Found y un breve mensaje de error.

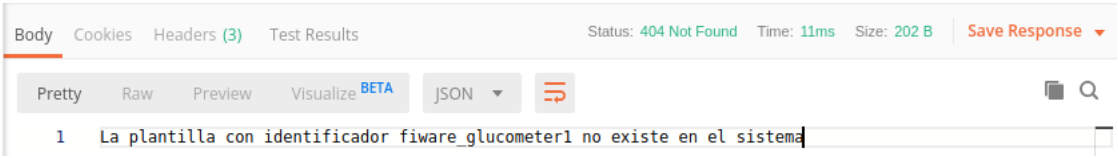

Figura 244. Respuestas en caso de que el DeviceMetric solicitado no exista

4.4.14 Recuperación de recurso Observation previamente creado

El gestor envía un mensaje GET a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la pasarela de forma local y en el puerto 9090. La aplicación espera la petición en la dirección "/observation/{identificador de la familia de dispositivos}".

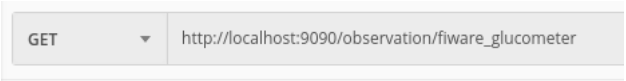

Figura 245. Petición recuperación Observation

La única cabecera HTTP utilizada es la "Accept" con valor "application/json", que indica que el cliente soporta una respuesta en formato JSON.

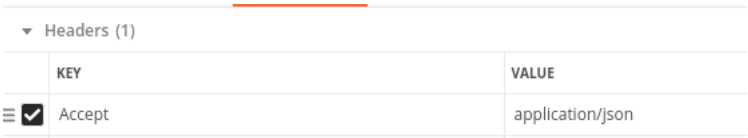

Figura 246. Cabecera HTTP recuperación Observation

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK y con el recurso Observation en el cuerpo de la respuesta codificado en JSON.

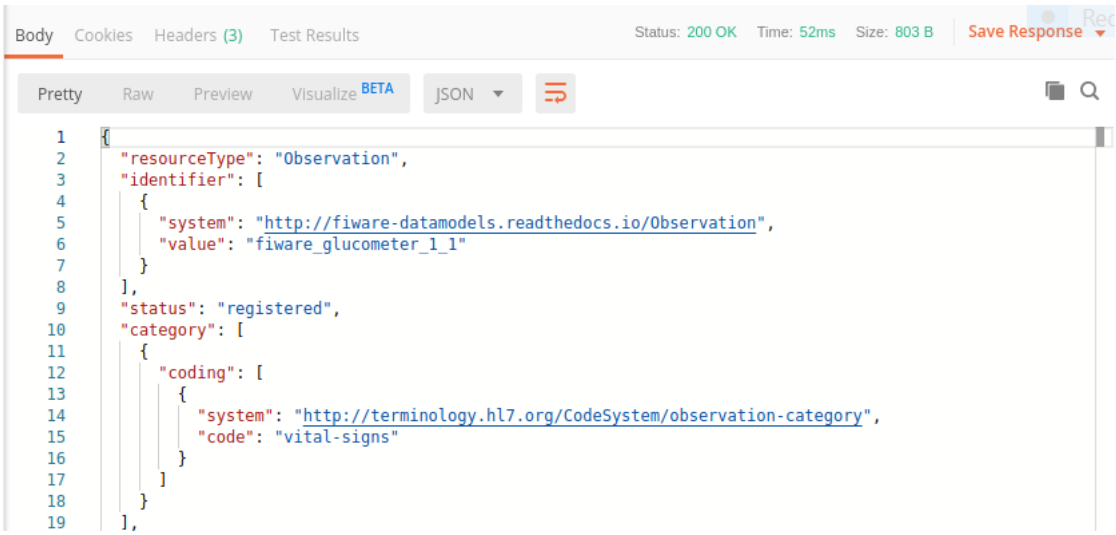

Figura 247. Fragmento del recurso Observation recuperado

Si el usuario solicita un recurso Observation con un identificador que no exista en la pasarela, el usuario recibe un mensaje 404 Not Found y un breve mensaje de error.

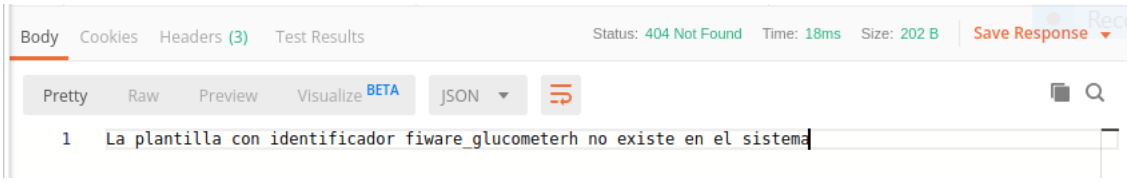

Figura 248. Respuesta en caso de que el Observation solicitado no exista

4.4.15 Actualización de un recurso DeviceDefinition previamente creado

El gestor envía un mensaje PUT a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la pasarela de forma local y en el puerto 9090. La aplicación espera la petición en la dirección "/devicedefinition/{identificador de la familia de dispositivos}".

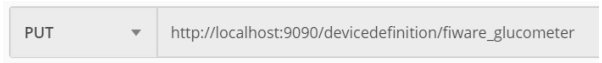

Figura 249. Petición actualización DeviceDefinition

Se usan dos cabeceras HTTP, la "Accept" con valor "application/json" que indica que el cliente soporta una respuesta en formato JSON. Y la "Content-Type" con el mismo valor, indicando el tipo de contenido del cuerpo de la petición.

Figura 250. Cabecera HTTP actualización DeviceDefinition

En el cuerpo del mensaje se encuentra el nuevo recurso DeviceDefinition en formato JSON siguiendo el estándar FHIR v4.0.0. En este caso demostrativo, se ha hecho una pequeña variación en un campo dentro de deviceName. Se ha cambiado el nombre de tipo manufacturer-name.

Figura 251. Nuevo recurso DeviceDefinition

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK.

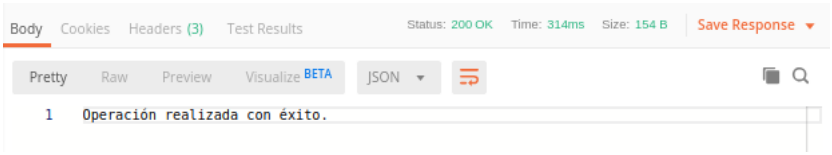

Figura 252. Respuesta exitosa actualización DeviceDefinition

Se comprueba que el recurso ha sido actualizado tanto en el servidor FHIR, como en nuestra base de datos, como en FIWARE.

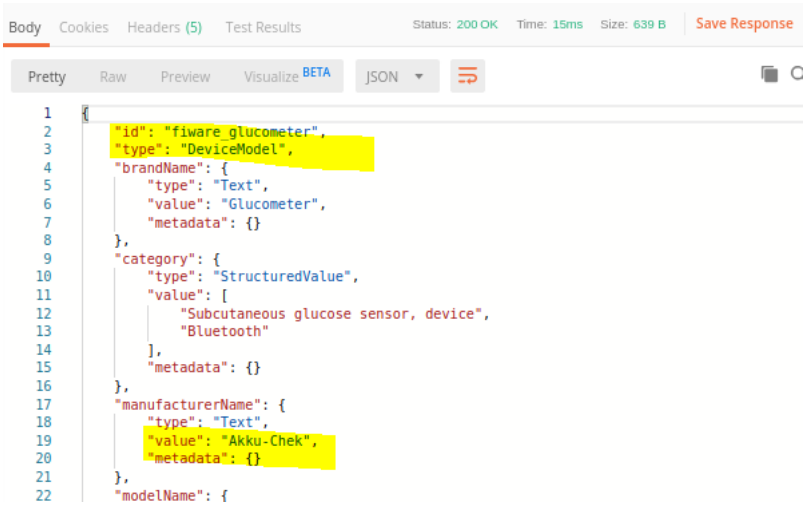

Figura 253. DeviceModel actualizado en Orion

HTTP 200 OK

Response Headers

```
X-Powered-By: HAPI FHIR 4.0.0 REST Server (FHIR Server; FHIR 4.0.0/R4)
Content-Type: application/fhir+xml;charset=utf-8
X-Request-ID: 630a88595054d145
```

Response Body


```
{
  "resourceType": "DeviceDefinition",
  "id": "1",
  "meta": {
    "versionId": "2",
    "lastUpdated": "2019-09-20T18:40:42.800+00:00"
  },
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/DeviceDefinition",
      "value": "fiware_glucometer"
    }
  ],
  "deviceName": [
    {
      "name": "Glucose sensor 789",
      "type": "model-name"
    },
    {
      "name": "Glucometer",
      "type": "user-friendly-name"
    },
    {
      "name": "Akku-Chek",
      "type": "manufacturer-name"
    }
  ],
  "modelNumber": "789",
  "type": {
    "coding": [
      {
        "system": "http://snomed.info/sct",
        "code": "701750003",
        "display": "Subcutaneous glucose sensor, device"
      }
```


Figura 254. Recurso DeviceDefinition actualizado en el servidor FHIR

Body Cookies Headers (3) Test Results Status: 200 OK Time: 55ms Size: 921 B Save Response

Pretty Raw Preview Visualize BETA JSON


```
{
  "resourceType": "DeviceDefinition",
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/DeviceDefinition",
      "value": "fiware_glucometer"
    }
  ],
  "deviceName": [
    {
      "name": "Glucose sensor 789",
      "type": "model-name"
    },
    {
      "name": "Glucometer",
      "type": "user-friendly-name"
    },
    {
      "name": "Akku-Chek",
      "type": "manufacturer-name"
    }
  ],
  "modelNumber": "789",
  "type": {
    "coding": [
      {
        "system": "http://snomed.info/sct",
        "code": "701750003",
        "display": "Subcutaneous glucose sensor, device"
      }
```


Figura 255. Recurso DeviceDefinition actualizado en al base de datos

Si el usuario solicita modificar un recurso DeviceDefinition con un identificador que no exista en la pasarela, el usuario recibe un mensaje 400 Bad Request y un breve mensaje de error.

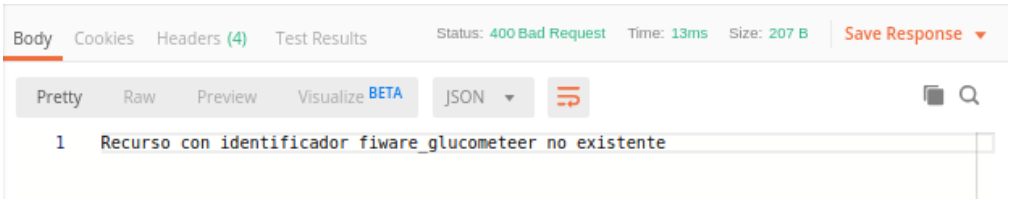

Figura 256. Respuesta en caso de que el DeviceDefinition no exista

Si el usuario envía a la pasarela un recurso DeviceDefinition con errores en la estructura,

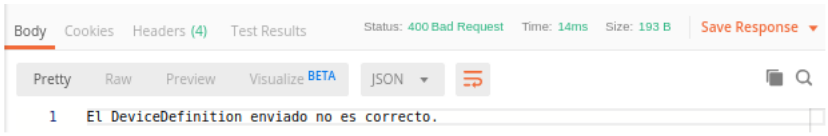

Figura 257. Respuesta en caso de que el DeviceDefinition presente errores

4.4.16 Actualización de un recurso Device previamente creado

El gestor envía un mensaje PUT a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la pasarela de forma local y en el puerto 9090. La aplicación espera la petición en la dirección "/device/{identificador del recurso Device}".

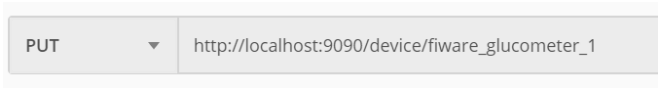

Figura 258. Petición actualización Device

Se usan dos cabeceras HTTP, la "Accept" con valor "application/json" que indica que el cliente soporta una respuesta en formato JSON. Y la "Content-Type" con el mismo valor, indicando el tipo de contenido del cuerpo de la petición.

Figura 259. Cabecera HTTP actualización Device

En el cuerpo del mensaje se encuentra el nuevo recurso Device en formato JSON siguiendo el estándar FHIR v4.0.0. En este caso demostrativo se ha hecho una pequeña variación en el campo serialNumber.

```
{
  "resourceType": "Device",
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/Device",
      "value": "fiware_glucometer_1"
    }
  ],
  "status": "active",
  "statusReason": [{"text":"online"}],
  "manufacturer": "Accu-Chek",
  "serialNumber":"1234567999",
  "type": {
    "coding": [
      {
        "system": "http://snomed.info/sct",
        "code": "701750003",
        "display": "Subcutaneous glucose sensor, device"
      }
    ]
  },
  "deviceName": [
    {
      "name":"Glucose sensor 789",
      "type":"model-name"
    }
  ],
  "modelNumber" : "789",
  "version" :{
    "type": {
     "text":"Hardware"
    },
    "value": "2"
  },
  "manufactureDate": "2015-08-08",
  "url": "192.168.1.254"
}
```


Figura 260. Nuevo recurso Device

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK.

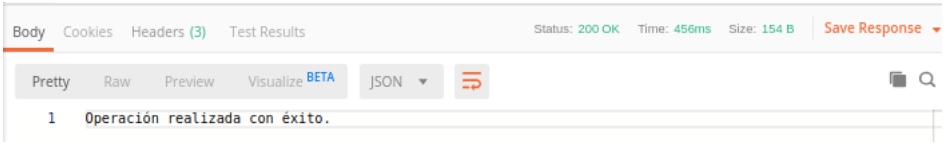

Figura 261. Respuesta exitosa de actualización Device

Se comprueba que el recurso ha sido actualizado tanto en el servidor FHIR, como en la base de datos, como en FIWARE.

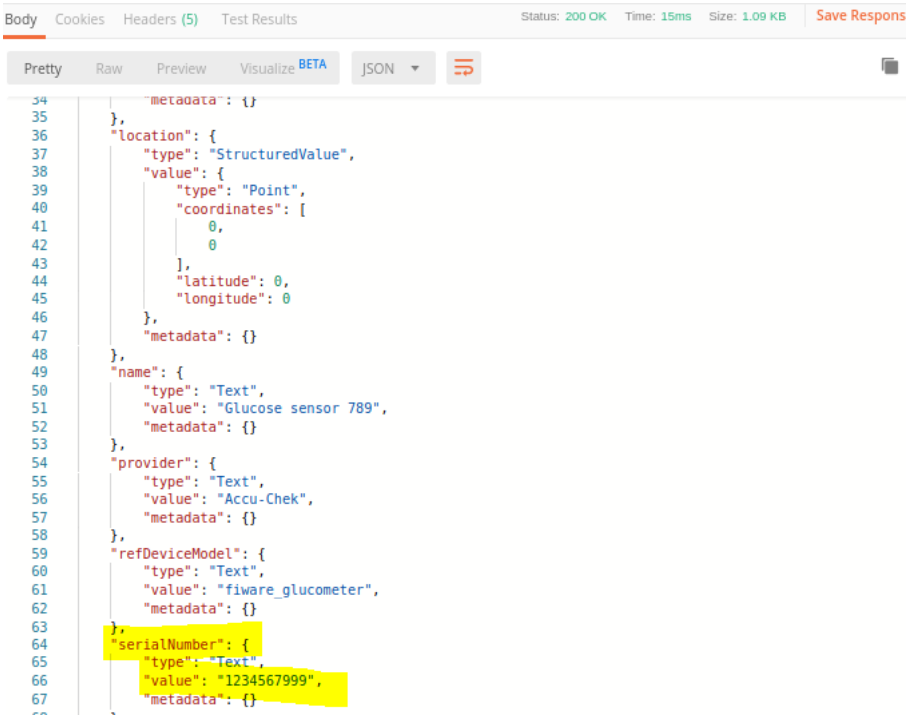

Figura 262. Recurso Device (FIWARE) actualizado en Orion

HTTP 200 OK

Response Headers

```
X-Powered-By: HAPI FHIR 4.0.0 REST Server (FHIR Server; FHIR 4.0.0/R4)
Content-Type: application/fhir+xml;charset=utf-8
X-Request-ID: 18d3ea3c7dc5a24e
```

Response Body


```
{
  "resourceType": "Device",
  "id": "3",
  "meta": {
    "versionId": "2",
    "lastUpdated": "2019-09-20T21:35:47.673+00:00"
  },
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/Device",
      "value": "fiware_glucometer_1"
    }
  ],
  "definition": {
    "reference": "DeviceDefinition/1"
  },
  "status": "active",
  "statusReason": [
    {
      "text": "online"
    }
  ],
  "manufacturer": "Accu-Chek",
  "manufactureDate": "2015-08-08",
  "serialNumber": "1234567999",
  "deviceName": [
    {
      "name": "Glucose sensor 789",
      "type": "model-name"
    }
  ],
  "modelNumber": "789",
  "type": {
    "coding": [
      {
        "system": "http://snomed.info/sct",
        "code": "701750003",
        "display": "Subcutaneous glucose sensor, device"
```


Figura 263. Recurso Device actualizado en el servidor FHIR

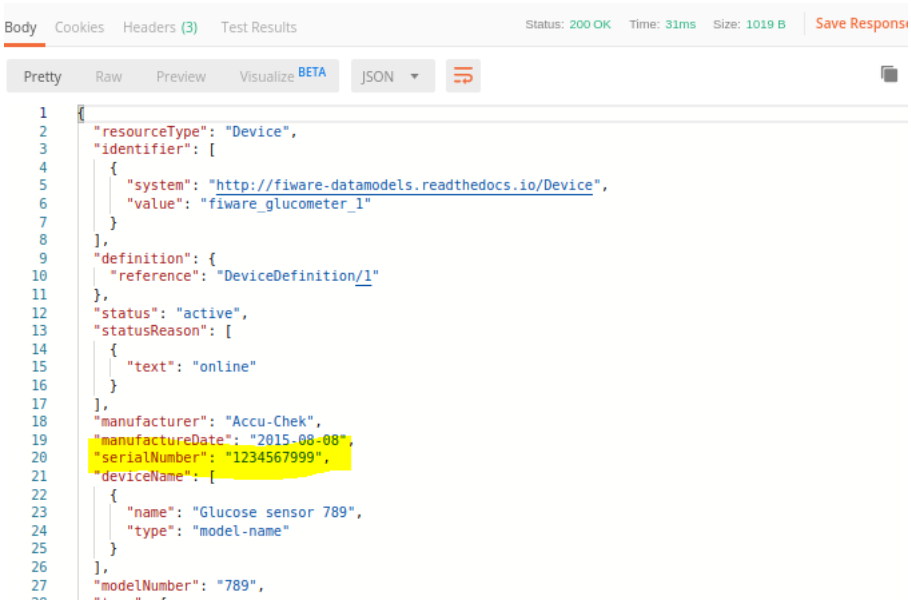

Figura 264. Recurso Device actualizado en la base de datos

Si el usuario solicita modificar un recurso Device con un identificador que no exista en la pasarela, el usuario recibe un mensaje 400 Bad Request y un breve mensaje de error.

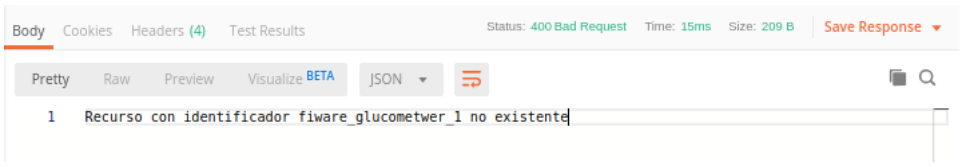

Figura 265. Respuesta en caso de que el Device no exista

Si el usuario manda un recurso Device a la pasarela con errores en la estructura, recibe un mensaje 400 Bad Request y un breve mensaje de error.

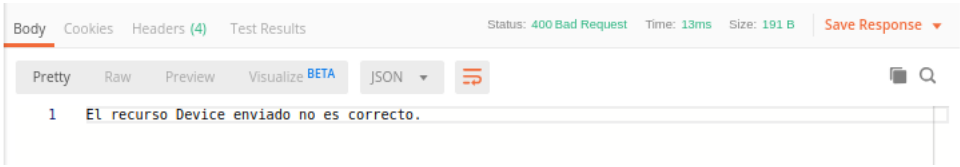

Figura 266. Respuesta en caso de que el Device presente errores

4.4.17 Actualización de un recurso DeviceMetric previamente creado

El gestor envía un mensaje PUT a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la aplicación Spring de forma local y en el puerto 9090. La aplicación espera la petición en la dirección "/devicemetric/{identificador de la familia de dispositivos}".

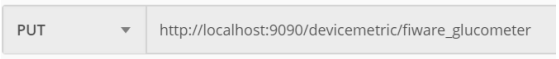

Figura 267. Petición actualización DeviceMetric

Se usan dos cabeceras HTTP, la "Accept" con valor "application/json" que indica que el cliente soporta una respuesta en formato JSON. Y la "Content-Type" con el mismo valor, indicando el tipo de contenido del cuerpo de la petición.

Figura 268. Cabecera HTTP actualización DeviceMetric

En el cuerpo del mensaje se encuentra el nuevo recurso DeviceMetric en formato JSON siguiendo el estándar FHIR v4.0.0. En este caso demostrativo se ha hecho una pequeña variación en un campo dentro de unit. Se ha cambiado ligeramente el campo display.

```
{
  "resourceType": "DeviceMetric",
  "identifier": [
    {
      "system": "http://fiware-datamodels.readthedocs.io/DeviceMetric",
      "value": "fiware_glucometer_1_1"
    }
  ],
  "type": {
    "coding": [
      {
        "system": "urn:iso:std:iso:11073:10101",
        "code": "150456"
      }
    ]
  },
  "unit": {
    "coding": [
      {
        "system": "urn:iso:std:iso:11073:10101",
        "code": "28716",
        "display": "MDC_CONC_GLU_ART_PRUEBA"
      }
    ]
  },
  "operationalStatus": "on",
  "category": "measurement",
  "measurementPeriod": {
    "repeat": {
      "frequency": 60,
      "periodUnit": "s"
    }
  },
  "calibration": [
    {
      "type": "unspecified",
      "state": "calibrated",
      "time": "2018-05-28T09:03:04-05:00"
    }
  ]
}
```


Figura 269. Nuevo DeviceMetric

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK.

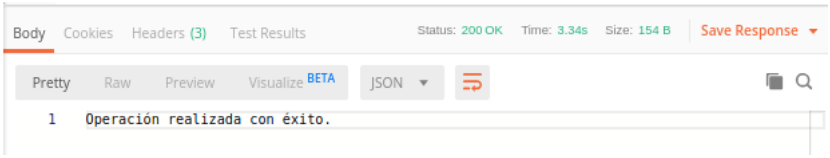

Figura 270. Respuesta exitosa de actualización DeviceMetric

Se comprueba que el recurso ha sido actualizado tanto en la base de datos, como en FIWARE.

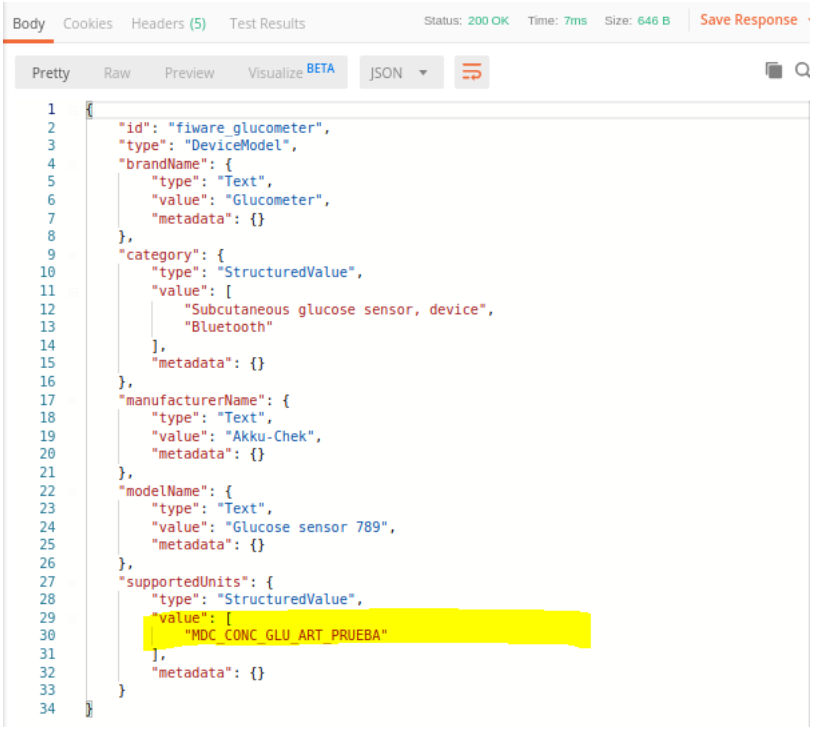

Figura 271. DeviceModel actualizado en Orion

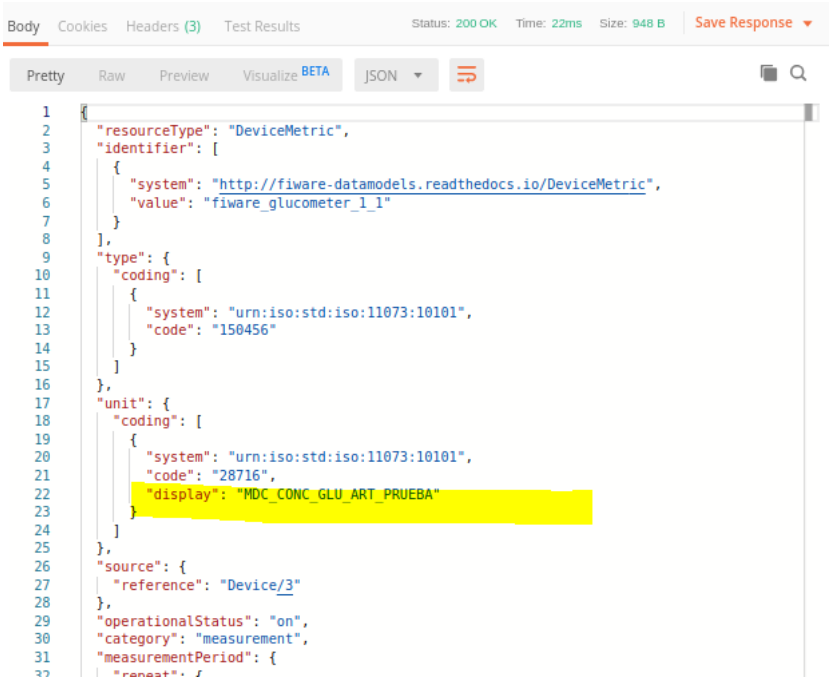

Figura 272. DeviceMetric actualizado en la base de datos

Si el usuario solicita modificar un recurso DeviceMetric con un identificador que no exista en la pasarela, el usuario recibe un mensaje 400 Bad Request y un breve mensaje de error.

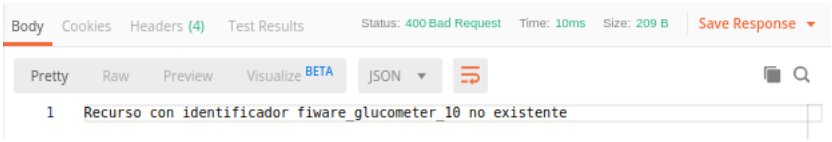

Figura 273. Respuesta en caso de que el DeviceMetric no exista

Si el usuario manda un recurso DeviceMetric a la pasarela con errores en la estructura, el usuario recibe un mensaje 400 Bad Request y un breve mensaje de error.

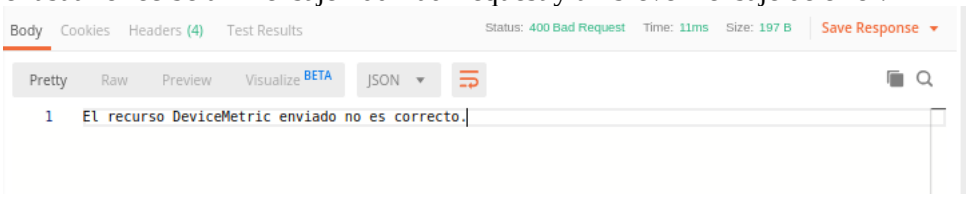

Figura 274. Respuesta en caso de que el DeviceMetric presente errores

4.4.18 Actualización de recurso Observation previamente creado

Se envía un mensaje PUT con las siguientes características:

Esta es la URL por defecto, ya que se ejecuta la aplicación Spring de forma local y en el puerto 9090. La aplicación espera la petición en la dirección "/updateobservation}".

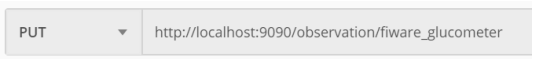

Figura 275. Petición actualización Observation

Se usan dos cabeceras HTTP, la “Accept” con valor “application/json” que indica que el cliente soporta una respuesta en formato JSON. Y la “Content-Type” con el mismo valor, indicando el tipo de contenido del cuerpo de la petición.

Figura 276. Cabecera actualización Observation

En el cuerpo del mensaje se encuentra el nuevo recurso Observation en formato JSON siguiendo el estándar FHIR v4.0.0. En este caso demostrativo se ha hecho una pequeña variación en un campo dentro de code. Se ha cambiado el nombre del atributo text.

Figura 277. Nuevo recurso Observation

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK.

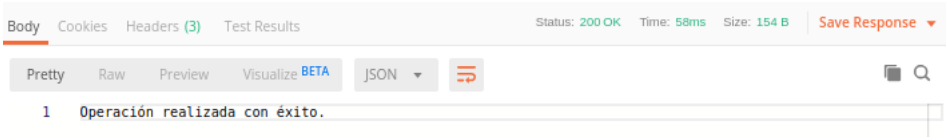

Figura 278. Respuesta exitosa actualización Observation

Se comprueba que el recurso ha sido actualizado en la base de datos.

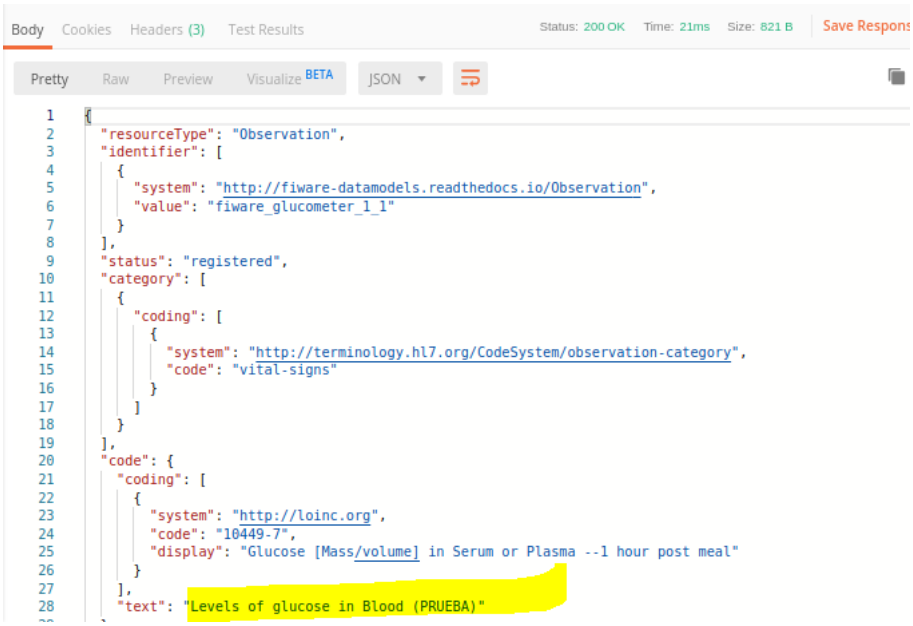

Figura 279. Recurso Observation actualizado en la base de datos

Si el usuario solicita modificar un recurso Observation con un identificador que no exista en la pasarela, el usuario recibe un mensaje 400 Bad Request y un breve mensaje de error.

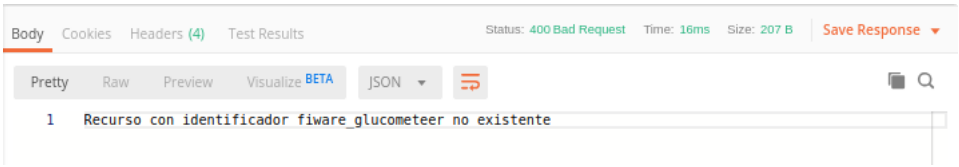

Figura 280. Respuesta en caso de que el Observation no exista

Si el usuario manda un recurso Observation a la pasarela con errores en la estructura, recibe un mensaje 400 BadRequest y un breve mensaje de error.

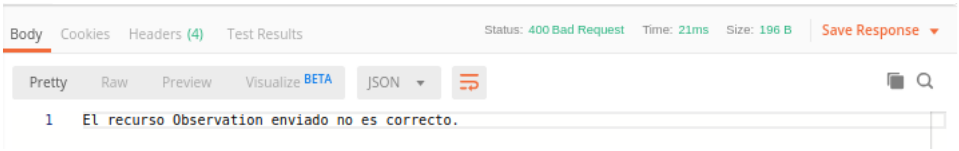

Figura 281. Respuesta en caso de que el Observation presente errores

4.4.19 Eliminación de una familia de dispositivos

El gestor envía un mensaje DELETE a la pasarela con las siguientes características.

Se usa la URL por defecto, ya que se ejecuta la pasarela de forma local y en el puerto 9090. La aplicación espera la petición en la dirección "/devicefamily/{id}", donde id se corresponde con el identificador de la familia a borrar.

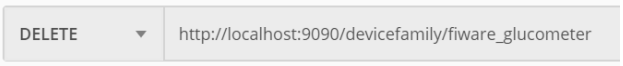

Figura 282. Petición eliminación familia de dispositivos

Este es el log de aplicación al procesar esta petición. Como se puede observar, se borran los 3 recursos Device en FIWARE y sus entradas asociadas en la base de datos. Además del DeviceModel de FIWARE y su entrada en la base de datos. También se borran las 5 suscripciones asociadas a esta familia.

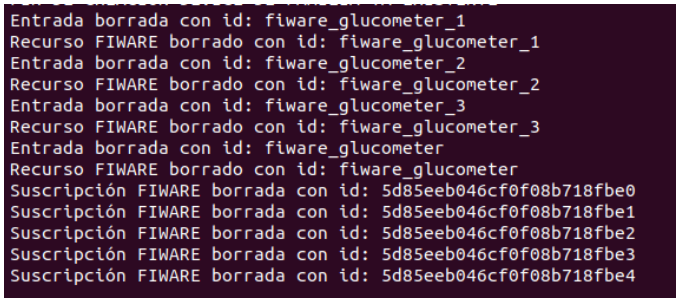

Figura 283. Log de la pasarela al borrar la familia de dispositivos

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con código 200 OK y un breve mensaje indicando el éxito de la operación.

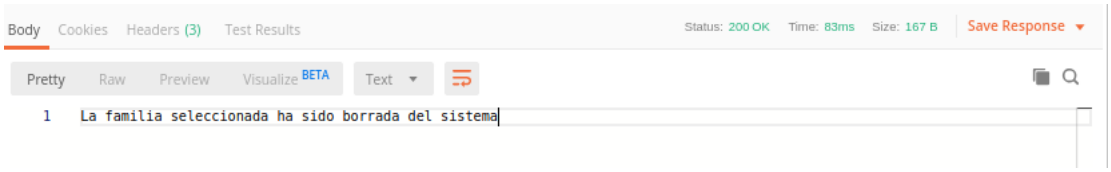

Figura 284. Respuesta en caso de éxito

Se comprueban que los recursos han sido borrados.

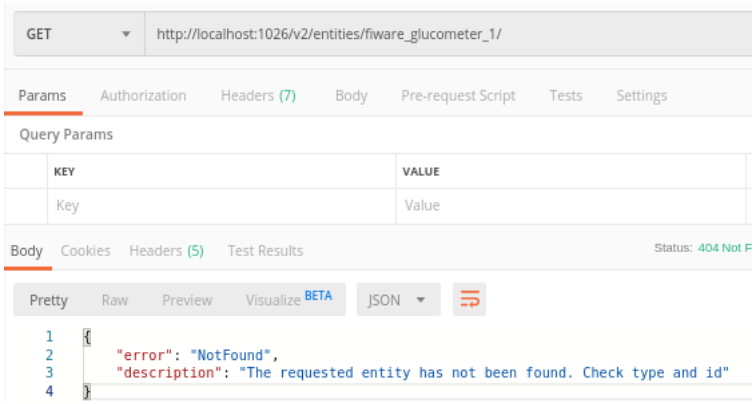

Figura 285. Recurso Device borrado de Orion

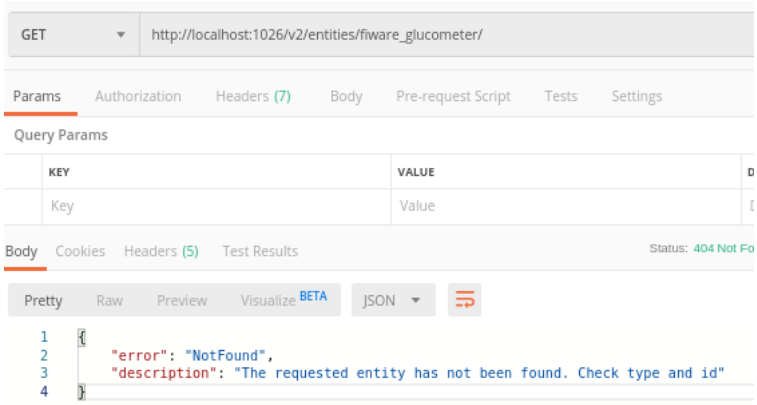

Figura 286. Recurso DeviceModel borrado de Orion

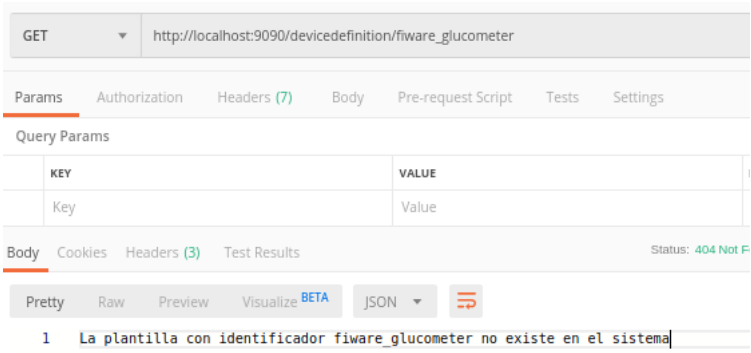

Figura 287. Recurso DeviceDefinition borrado de la pasarela

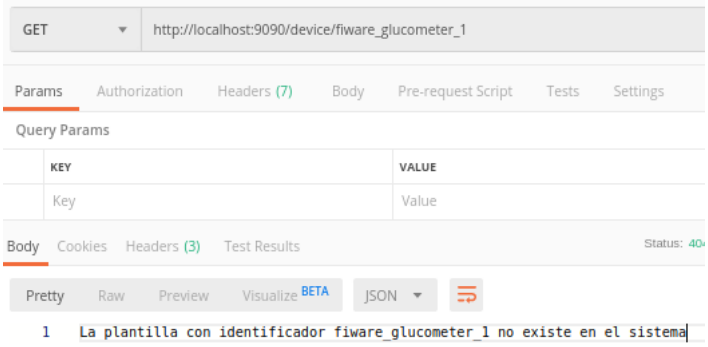

Figura 288. Recurso Device borrado de la pasarela

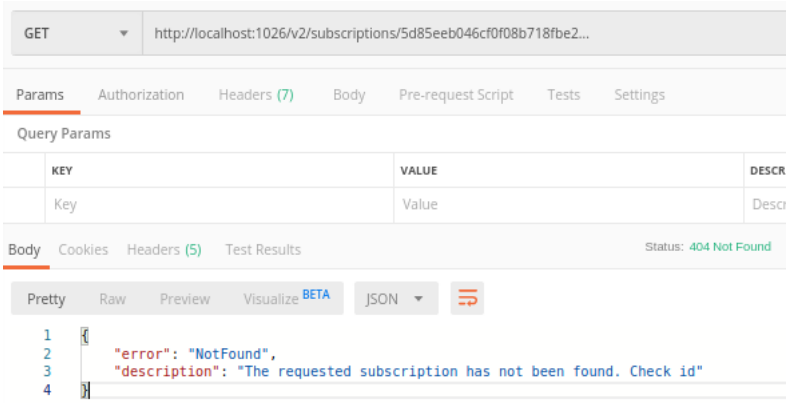

Figura 289. Suscripción borrada de Orion

Si el usuario solicita borrar una familia con un identificador que no exista en la pasarela, el usuario recibe un mensaje 400 Bad Request y un breve mensaje de error.

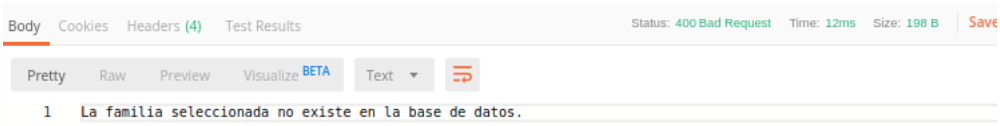

Figura 290. Respuesta si la familia seleccionada no existe

4.4.20 Eliminación de un dispositivo concreto

El gestor envía un mensaje DELETE a la pasarela con las siguientes características.

Se usa es la URL por defecto, ya que se ejecuta la pasarela de forma local y en el puerto 9090. La aplicación espera la petición en la dirección "/device/{id}", donde id se corresponde con el identificador del dispositivo a borrar.

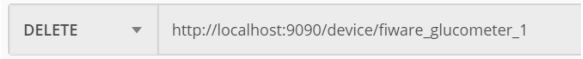

Figura 291. Petición eliminación Device

Este es el log de aplicación SpringBoot al procesar esta petición. Como se puede observar, se borra el recurso solicitado tanto de FIWARE como de la base de datos. Además, se elimina de la lista de dispositivos asociados a su familia.

```
Entrada de la base de datos borrada con id: fiware_glucometer_2
Recurso FIWARE borrado con id: fiware_glucometer_2
Id base: fiware_glucometer
Plantilla recuperada
Elemento borrado de la lista con id: fiware_glucometer_2
Plantilla guardada en la BD
```

Figura 292. Log de la pasarela eliminación Device

Si todo va bien, la pasarela responde al usuario con un mensaje HTTP con Código 200 OK y un breve mensaje indicando el éxito de la operación.

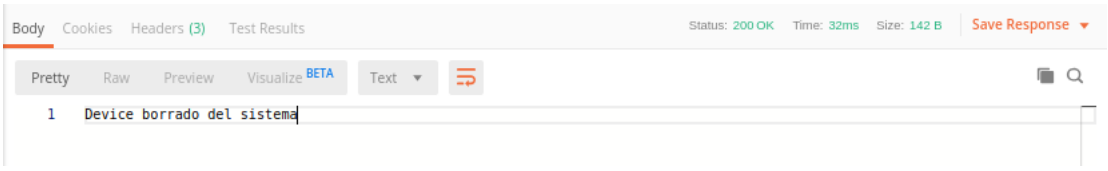

Figura 293. Respuesta exitosa eliminación Device

Se comprueba que los recursos han sido borrados.

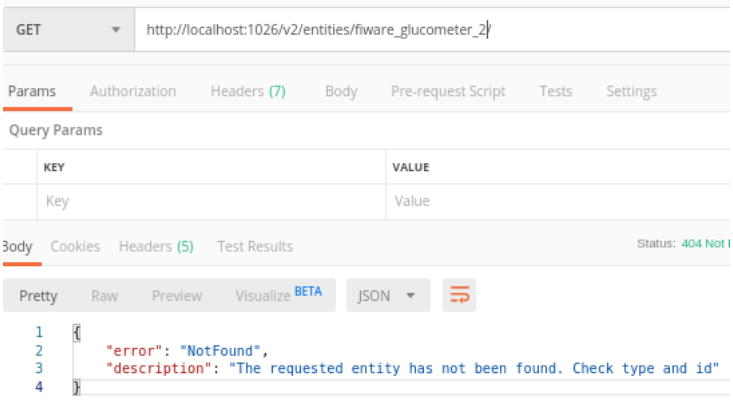

Figura 294. Recurso Device (FIWARE) borrado de Orion

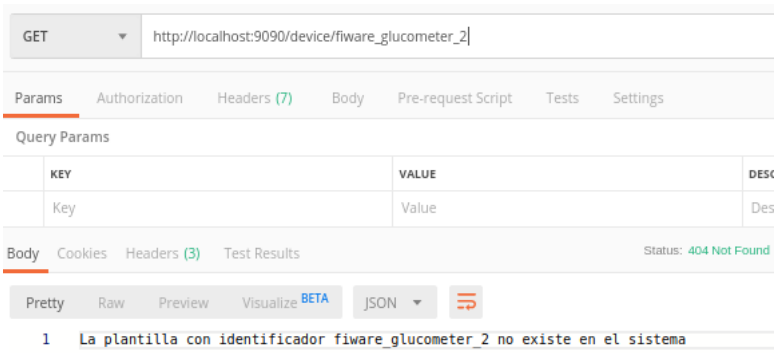

Figura 295. Recurso Device (FHIR) borrado del servidor FHIR

Si el usuario solicita borrar un recurso Device con un identificador que no exista la pasarela, el usuario recibe un mensaje 400 Bad Request y un breve mensaje de error.

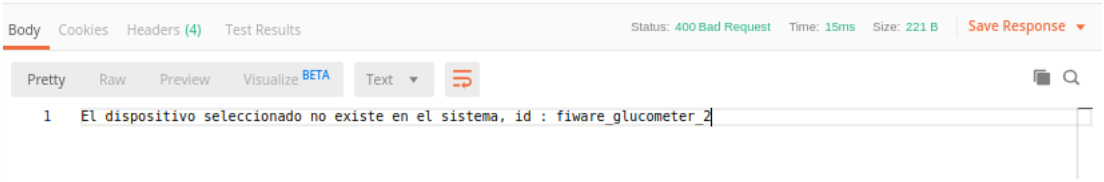

Figura 296. Respuesta si el Device seleccionado no existe

5 CONCLUSIONES

En este capítulo final se tratan las principales dificultades superadas a lo largo del proyecto. Además, se analizan las posibles líneas futuras de desarrollo, para ampliar y mejorar la aplicación desarrollada. Por último, se finaliza con una conclusión personal.

5.1 Esfuerzos necesarios

- **Estándar HL7 FHIR:** Este estándar es uno de los pilares del proyecto. Este primer contacto con un sistema sanitario ha implicado un análisis profundo de los recursos utilizados. Se han realizado numerosas lecturas y consultas de la especificación a lo largo del trabajo para la asimilación correcta de los conceptos. Además, la implementación del estándar usada, HAPI FHIR, ha supuesto todo un reto. En ocasiones, la documentación no contaba con todos los ejemplos necesarios para su correcta comprensión. Esto ocurría por ejemplo a la hora de crea un cliente FHIR basado en anotaciones.

- **Estándar NGSIv2:** El otro pilar fundamental es el estándar NGSIv2 en el que se basa el Orion Context Broker. Se han estudiado en profundidad los conceptos de entidad, suscripción y notificación. También se ha profundizado en su interfaz REST para el manejo de las entidades y suscripciones. Se ha tenido que crear un cliente REST completo ya que no se ha encontrado ninguno, ya hecho en java, que sea compatible con la versión actual.

- **Correspondencia FHIR-FIWARE:** Esta tarea ha requerido bastante tiempo, debido a la gran complejidad de los recursos del estándar FHIR. Además de las múltiples diferencias con los recursos del estándar FIWARE:

- **Aprendizaje del framework Spring de Java:** Se estableció desde un principio la utilización del lenguaje java, ya que la librería HAPI FHIR propuesta está programada en este lenguaje. Al ver las necesidades del proyecto, se decidió que este framework facilitaba enormemente las tareas necesarias. Desde el despliegue de la aplicación, hasta la creación de la interfaz REST o el cliente REST NGSIv2. Ha sido un aprendizaje desde cero, no obstante, la documentación de este framework es muy buena.

- **Docker:** Esta tecnología ha agilizado el despliegue de los componentes del escenario final. También ha sido un aprendizaje desde cero. Pero la calidad de la documentación oficial y de los diversos tutoriales de internet han permitido adquirir conocimientos básicos para la implementación.

5.2 Línea futura de desarrollo

Este proyecto es el primer paso para unir las plataformas FHIR y FIWARE. En el caso de la implementación de esta pasarela en un entorno real, existen una serie de mejoras y tareas que habría que desarrollar:

- **Mecanismo de seguridad:** Es fundamental en el uso real del servicio ya que se intercambia información sanitaria. FHIR ni es un protocolo de seguridad, ni define ninguna funcionalidad relacionada. No obstante, si define los protocolos [36] que deben utilizarse:

    1. Sincronización de relojes mediante NTP/SNTP.
    2. Intercambio de datos mediante TLS/SSL. Por ejemplo, https.
    3. Sistemas de autenticación de usuarios/clientes. Se recomienda el estándar OAuth [37].
    4. Sistemas de autorización y control de acceso.
    5. Recursos de auditoría de eventos y de procedencia.
    6. Uso de firmas digitales.
    7. Precaución en los archivos adjuntos.
    8. Uso de etiquetas relacionadas con la seguridad.
    9. Políticas de gestión de datos.
    10. Precaución a la hora de mostrar la narrativa de los recursos FHIR.
    11. Validación de los datos de entrada.

- **Interfaz gráfica para gestionar la aplicación:** Se podría crear una aplicación web, que usara la interfaz REST ofrecida por la pasarela, para facilitar los trabajos de gestión.

- **Tratamiento de los datos:** Se podrían explotar los datos obtenidos de los sensores de forma inteligente. Actualmente, la pasarela únicamente genera observaciones con los datos medidos. No se aprovechan después para ningún fin.

- **Aumento de funcionalidades:** En este proyecto se han utilizado un número reducido de recursos de ambas plataformas. Se podría incluir recursos relacionados con los pacientes que usan la plataforma, y con las empresas que fabrican los dispositivos.

5.3 Conclusión

El desarrollo del proyecto me ha servido para profundizar los conocimientos adquiridos a lo largo de la carrera, especialmente en el concepto de servicio REST. Hoy en día, es fundamental que los desarrolladores tengan una clara comprensión de esta tecnología. Especialmente, con la fuerza que llevan cogiendo los últimos años los servicios cloud que usan extensamente esta tecnología.

La codificación del sistema me ha dado lo oportunidad de refrescar conocimientos del lenguaje java estudiado en cursos anteriores. Además, me ha permitido el aprendizaje del framework Spring. Es posiblemente el framework más utilizado, por lo que muchas empresas grandes lo valoran. También he adquirido algunos conocimientos de la tecnología Docker, que está muy de moda en el despliegue de servicios en la nube.

Además, entender la importancia de las TICS aplicadas en el sector de la salud, me ha abierto la mente a nuevas líneas de trabajo que no había considerado con anterioridad.

6 REFERENCIAS

[1] [En línea]. Available: https://empresas.blogthinkbig.com/fiware-el-estandar-que-necesita-el-iot/. [Último acceso: 22 09 2019].

[2] [En línea]. Available: https://searchhealthit.techtarget.com/definition/FHIR-Fast-Healthcare-Interoperability-Resources. [Último acceso: 22 9 2019].

[3] [En línea]. Available: https://www.hl7.org/about/index.cfm. [Último acceso: 26 10 2019].

[4] [En línea]. Available: https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html. [Último acceso: 22 9 2019].

[5] [En línea]. Available: https://github.com/telefonicaid/fiware-orion/blob/master/README.md. [Último acceso: 22 09 2019].

[6] [En línea]. Available: https://fiware.github.io/specifications/ngsiv2/stable/. [Último acceso: 22 09 2019].

[7] [En línea]. Available: https://www.mongodb.com/what-is-mongodb?lang=es-es. [Último acceso: 22 9 2019].

[8] [En línea]. Available: https://www.hl7.org/fhir/device.html. [Último acceso: 22 09 2019].

[9] [En línea]. Available: https://www.hl7.org/fhir/devicedefinition.html. [Último acceso: 22 9 2019].

[10] [En línea]. Available: https://www.hl7.org/fhir/devicemetric.html. [Último acceso: 22 9 2019].

[11] [En línea]. Available: https://www.hl7.org/fhir/observation.html#Observation. [Último acceso: 22 9 2019].

[12] [En línea]. Available: https://tools.ietf.org/html/rfc7231. [Último acceso: 22 9 2019].

[13] [En línea]. Available: https://www.json.org/. [Último acceso: 22 9 2019].

[14] [En línea]. Available: https://en.wikipedia.org/wiki/Eclipse_(software). [Último acceso: 1 10 2019].

[15] [En línea]. Available: https://www.getpostman.com/. [Último acceso: 1 10 2019].

[16] [En línea]. Available: https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html. [Último acceso: 1 10 2019].

[17] [En línea]. Available: https://gsuite.google.com/marketplace/app/drawio_diagrams/671128082532. [Último acceso: 1 10 2019].

[18] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Java_(programming_language). [Último acceso: 22 9 2019].

[19] [En línea]. Available: http://hapifhir.io/. [Último acceso: 22 9 2019].

[20] [En línea]. Available: https://curiotek.com/2017/06/16/java-que-es-spring/. [Último acceso: 22 9 2019].

[21] [En línea]. Available: https://dzone.com/articles/what-is-spring-boot. [Último acceso: 22 9 2019].

[22] [En línea]. Available: https://es.wikipedia.org/wiki/Maven. [Último acceso: 22 09 2019].

[23] [En línea]. Available: http://www.georss.org/. [Último acceso: 30 09 2019].

[24] [En línea]. Available: https://macwright.org/2015/03/23/geojson-second-bite.html. [Último acceso: 26 10 2019].

[25] [En línea]. Available: https://fiware-datamodels.readthedocs.io/en/latest/index.html. [Último acceso: 22 9 2019].

[26] [En línea]. Available: https://fiware-datamodels.readthedocs.io/en/latest/Device/Device/doc/spec/index.html. [Último acceso: 22 09 2019].

[27] [En línea]. Available: https://fiware-datamodels.readthedocs.io/en/latest/Device/DeviceModel/doc/spec/index.html. [Último acceso: 22 09 2019].

[28] [En línea]. Available: https://www.fda.gov/medical-devices/device-advice-comprehensive-regulatory-assistance/unique-device-identification-system-udi-system. [Último acceso: 04 10 2019].

[29] [En línea]. Available: https://aws.amazon.com/es/docker/. [Último acceso: 22 9 2019].

[30] [En línea]. Available: https://en.wikipedia.org/wiki/Automatic_identification_and_data_capture. [Último acceso: 05 10 2019].

[31] [En línea]. Available: https://en.wikipedia.org/wiki/Code_of_Federal_Regulations. [Último acceso: 05 10 2019].

[32] [En línea]. Available: https://loinc.org/. [Último acceso: 05 10 2019].

[33] [En línea]. Available: https://standards.ieee.org/standard/11073-10101-2019.html. [Último acceso: 05 10 2019].

[34] [En línea]. Available: https://github.com/FIWARE/data-models. [Último acceso: 15 10 2019].

[35] [En línea]. Available: https://hub.docker.com. [Último acceso: 05 10 2019].

[36] [En línea]. Available: https://www.hl7.org/fhir/security.html. [Último acceso: 21 10 2019].

[37] [En línea]. Available: https://oauth.net/. [Último acceso: 21 10 2019].

[38] «eSalud, web de referencia sobre la eHealth en español,» [En línea]. Available: https://laesalud.com/esalud/. [Último acceso: 21 9 2019].